Summer 1992

Multigrid Algorithms for Massively Parallel Machines

Satyanarayan Gupta
Old Dominion University

Multigrid Algorithms for Massively Parallel Machines

by

Satyanarayan Gupta
B.E. June 1983, Birla Institute of Technology and Science, India
M.Tech June 1988, Indian Institute of Technology, India

A Dissertation submitted to the Faculty of Old Dominion University in Partial Fulfillment of the Requirement for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY

August, 1992

Approved by :

Mohammad Zubair, advisor

Chester E. Grosch, advisor

Kurt Maly

Thomas V. Jackson

Abstract

MULTIGRID ALGORITHMS FOR MASSIVELY PARALLEL MACHINES

Satyanarayan Gupta
Old Dominion University, 1992
Advisors : M. Zubair and C.E. Grosch

Numerical solutions of partial differential equations (*pde*'s) are required in many physical problems arising in areas such as computational fluid dynamics, atmospheric sciences, electromagnetics etc.. One of the most popular methods of solving *pde*'s is the use of the multigrid algorithm. However, the implementation of the multigrid algorithm on massively parallel machines is not very efficient because of (i) low processor utilization and (ii) high communication overheads. These problems need to be addressed to make better use of massively parallel machines for solving *pde*'s using the multigrid algorithm.

In this dissertation, we present three parallel multigrid algorithms which address the above mentioned problems and thus obtain a better performance on massively parallel machines than the standard multigrid algorithm. The first of these, the *Overlap Parallel Multigrid (OPMG)* algorithm, uses unutilized processors on the coarse grids of the multigrid hierarchy to do additional computation. The additional computation improves the convergence rate of the multigrid algorithm and thus reduces the total parallel execution time to solve a problem. The second algorithm, the *Chopped Parallel Multigrid (CPMG)* algorithm, reduces the computational work on the coarse grids of the multigrid hierarchy, while keeping the convergence rate per

cycle almost the same. The reduction in the computational work reduces the average parallel execution time per cycle, which in turn results in a reduced total parallel execution time. A combination of the complementary approaches used by these two algorithms is the source for our third algorithm, the hybrid algorithm. The hybrid algorithm obtains a better performance than the standard multigrid algorithm by improving the convergence rate per cycle and also by reducing the average parallel execution time per cycle. Both these factors reduce the total parallel execution time for solving *pde*'s using the multigrid algorithm.

We implemented the above three algorithms and also the standard multigrid algorithm on a massively parallel SIMD machine, the AMT-DAP/510, consisting of 1024 processors. The parallel implementation results show that our algorithms obtain a significant advantage over the standard multigrid algorithm. On the average, a speed-up of approximately 30%, 40% and 60% over the standard multigrid algorithm is obtained by the *OPMG* algorithm, the *CPMG* algorithm and the hybrid algorithm respectively.

Acknowledgements

I would like to express my sincere gratitude to a number of people for their support during my dissertation work. First of all, I would like to thank my advisors, Dr. Mohammad Zubair and Prof. Chester Grosch for introducing me to the exciting field of parallel scientific computing and for guiding me throughout this effort with their deep insight. I am indebted to them for giving me encouragement and moral support during the last four years. I would also like to thank Prof. Kurt Maly and Dr. Thomas Jackson for their support and suggestions as my thesis committee members.

My special thanks to Dr. M. Yousuff Hussaini and Dr. Joel Saltz of the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA, for providing me the opportunity to work with leading scientists in the area of parallel scientific computing.

I thank all the computer science faculty, my fellow graduate students and computer science office staff for making my stay at Old Dominion University a memorable one. Special thanks to Mary Merritt, Gary Carlson and Julia Tressel for their help in preparing this manuscript.

I would like to thank my parents for their encouragement and support in fulfilling my academic dreams. It was the thought of their sacrifices and love for me which kept me going through the hard times during the course of this work.

Last and most important, I would like to thank my wife Madhurima for her support and the sacrifices she has made during my dissertation work.

Contents

List of Figures

List of Tables

Nomenclature

G^i	ith grid in hierarchy of grids
L^i	Discretization of the differential operator on G^i
P^i	Projection operator which projects values from G^{i+1} to G^i
I^i	Interpolation operator which interpolates values from G^{i-1} to G^i
S^i	Iteration matrix for G^i
v^i	Approximate solution on G^i
f^i	Forcing function on G^i
r^i	Residual on G^i
w_k	kth eigenvector of matrix
e_k^i	kth Fourier mode of error on G^i
ptu	Parallel execution time for one cycle of V-cycle multigrid algorithm
λ_k	kth eigenvalue of matrix
β	Average reduction in residual norm per cycle
γ	Average reduction in residual norm per ptu

Chapter 1

Introduction

1.1 Introduction

Numerical solutions of partial differential equations (*pde's*) are required in many physical problems arising in areas such as computational fluid dynamics, atmospheric sciences, electromagnetics etc.. Typically, these *pde*'s are solved by discretizing the problem domain using a grid. A discrete solution of a *pde* on the grid points is obtained by using relaxation schemes such as weighted Jacobi and Gauss-Seidel schemes. These relaxation schemes start with an initial approximate solution and iteratively improve it, until it converges to the exact solution within a desired level of accuracy. The convergence rate of these relaxation schemes drops significantly after a few initial iterations. The multigrid technique [10, 11, 1, 19, 29] accelerates the convergence of these relaxation schemes by using a hierarchy of coarser grids in

addition to the grid on which solution is sought. A coarse grid in a multigrid hierarchy is usually formed by choosing alternate grid points of the next finer grid. A detailed description of the multigrid technique is given in Chapter 2. The multigrid technique is also referred to as the multigrid algorithm. Throughout this dissertation we will use the terms multigrid technique and multigrid algorithm interchangeably.

The implementation of the multigrid algorithm on massively parallel machines consisting of thousands of processors presents two major problems as the grids in the hierarchy become coarser, namely (i) lower processor utilization and (ii) higher communication cost. The processor utilization is defined as the ratio of the number of processors doing useful work to the total number of processors. The communication cost is part of the total execution time spent in obtaining values from the other processors. For a better explanation of these problems, we consider an example with a hierarchy of 2-D grids in which the finest grid has as many points as the number of processors in a mesh connected parallel machine. A straightforward mapping of two grids of this hierarchy is shown in Figure 1.1.

The computation in the multigrid algorithm consists of iterations of the relaxation scheme used and some other operations to transfer data between adjacent grids of the multigrid hierarchy. In most of these operations, the computation on each grid point requires values from its neighboring grid points. For the finest grid, all the processors have one grid point mapped onto them, and therefore all of them will be used during computation. The communication cost on this grid is also minimal because the neighboring grid points are mapped onto neighboring processors.

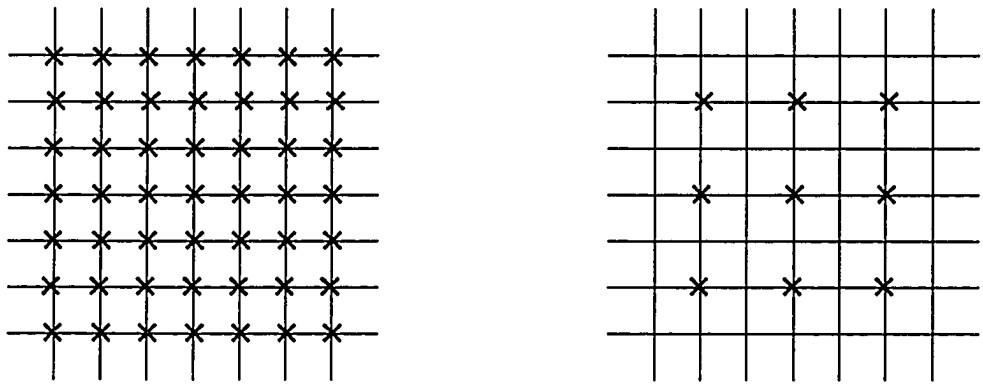


Figure 1.1 : Mapping of two grids onto a processor array

On the other hand, for the next coarse grid only one fourth of the processors have grid points mapped onto them, which gives a processor utilization of only 25% for this grid. The communication cost is also higher on this grid because the neighboring grid points are mapped onto processors two distances apart. In Figure 1.2 and Figure 1.3 we give quantitative results for processor utilization and communication overheads for a five grid hierarchy mapped onto a 1024 processor, mesh connected parallel machine. The finest grid in this case has 32×32 grid points. In these figures, grid 5 is the finest grid and grid 1 is the coarsest grid. These results show that although the processor utilization for the finest grid is 100%, it drops down to 0.3% for the coarsest grid. Similarly, the relative communication cost on the coarsest grid is 5.35 times more than the finest grid. The relative communication cost is computed by taking the ratio of the communication cost on a grid to the communication cost on the finest grid. The severe nature of these problems suggests that it is necessary to address them to make better use of parallel machines for the implementation of the multigrid algorithm.

It must be noted that these problems of communication overheads and low processor utilization are of importance only when the number of processors in the parallel machine are of the same order as the number of grid points in the finest grid. For those cases which have a greater number of grid points than processors, these problems arise only for the grids where the number of grid points is less than the number of processors.

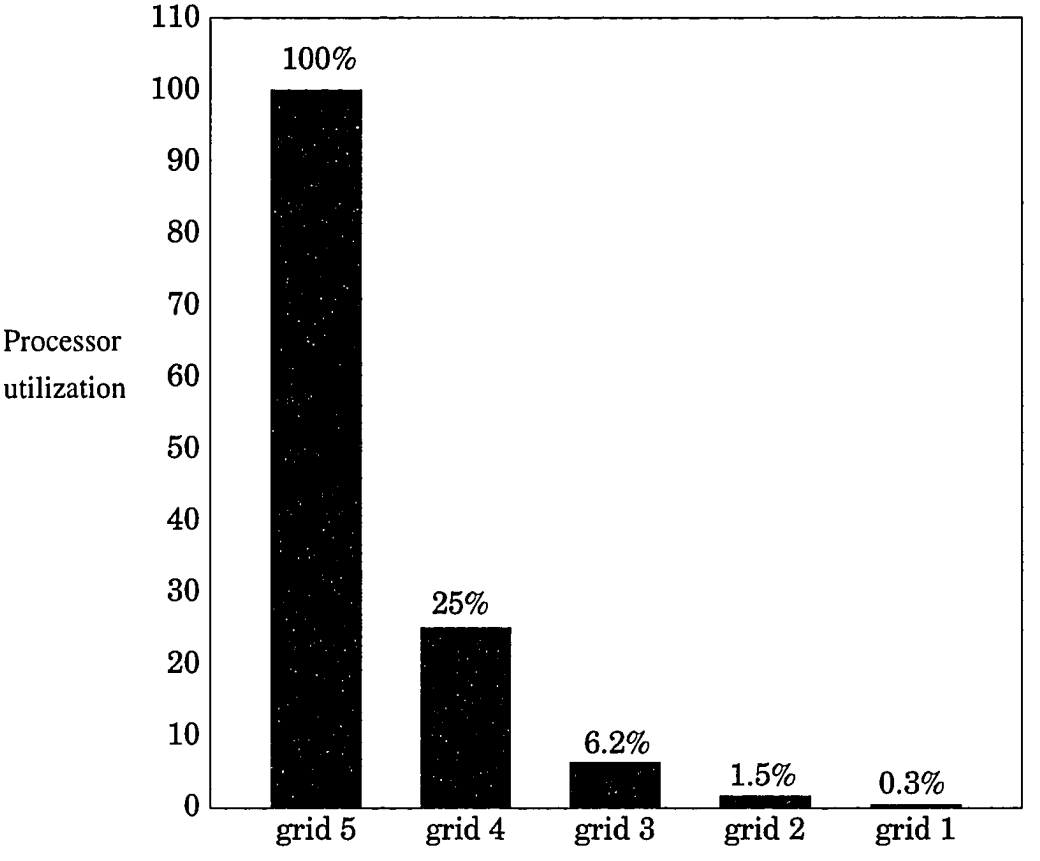


Figure 1.2 : Processor utilization for a 1024 processor machine

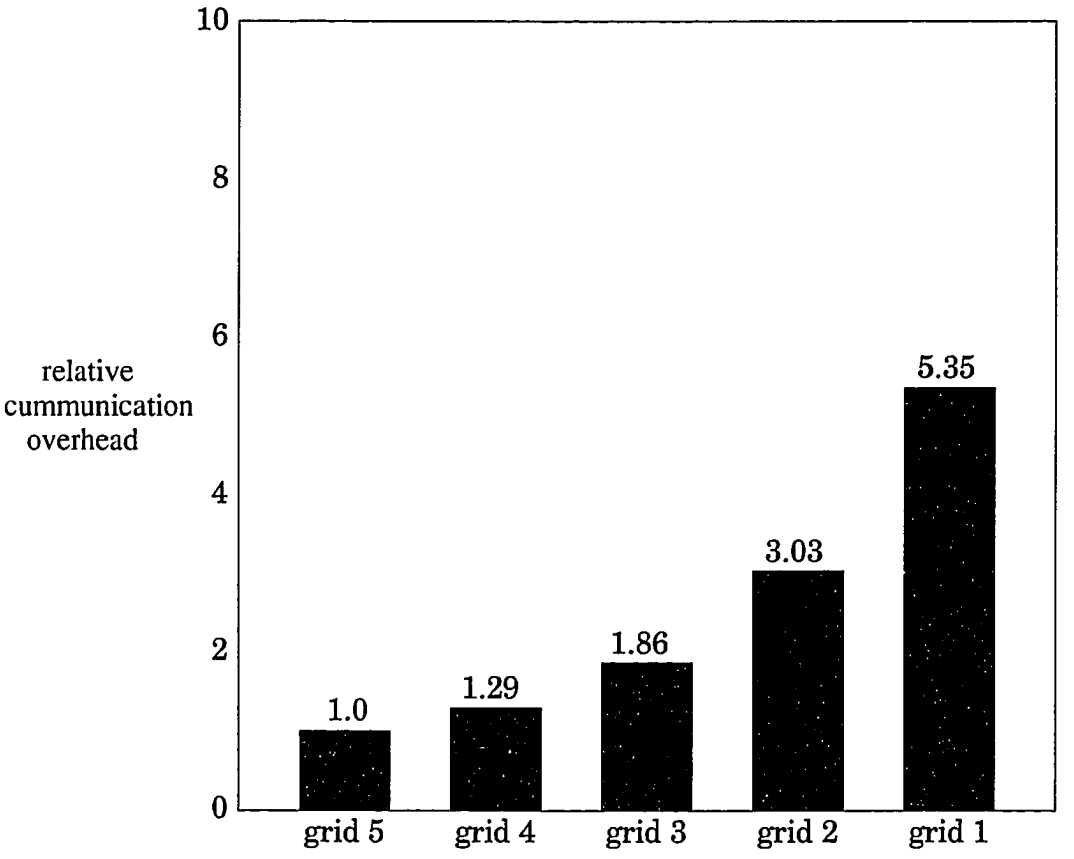


Figure 1.3 : Communication overheads for a 1024 processors machine

1.2 Previous Work

The previous work done by various researchers to address the problems associated with the parallel implementation of the multigrid algorithm can be classified into two broad categories. The work in the first category deals with adapting parallel architectures to implement the multigrid algorithm. The development of various mapping and implementation strategies on a variety of parallel computer architectures falls under this category. The work in the second category deals with adapting the multigrid algorithm to alleviate the previously mentioned problems on parallel machines. Based on the approach used, this category can be further divided as shown in Figure 1.4. In the first approach of this category, the unutilized processors on the coarse grids are used to do extra computational work. This extra work is generated by modifying the multigrid algorithm and improves the convergence rate. Since the additional work is done on unutilized processors, the improved convergence rate should not increase computational cost. In the second approach, the number of unutilized processors is decreased by reducing the computation in those steps which cause low processor utilization. Since computation on coarse grids causes low processor utilization and high communication overheads, reduction in computational work on these grids addresses both of these problems effectively. However, the reduced work in this approach should not degrade the convergence rate of the algorithm.

An excellent overview of the multigrid algorithms for solving the problems of processor utilization and communication overheads on parallel machines is given by

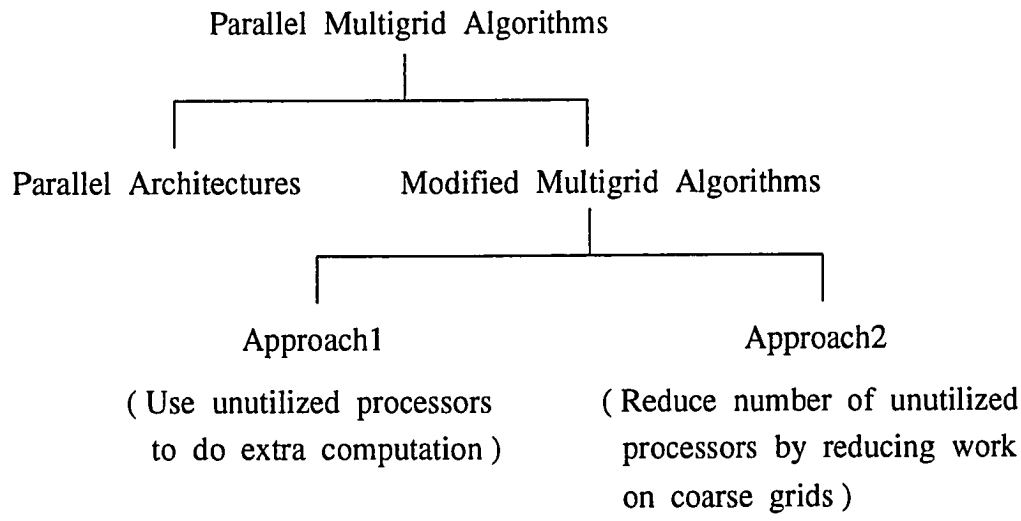


Figure 1.4 : Classification of parallel multigrid algorithms

Chan and Tuminaro [5]. Here, we discuss some of these works in the context of our classification.

1.2.1 Parallel Architectures

Researchers have suggested various parallel architectures for the implementation of the multigrid algorithm. The main feature of these architectures is that they try to reduce communication overheads during computation on the coarse grids. Here, we give a brief description of the work done in this area.

Grosch

The problem of high communication overheads on coarse grids is addressed by Grosch [18] by using the Perfect Shuffle Nearest Neighbor architecture (PSNN)[28]. The PSNN architecture can be defined as follows. For an n processor PSNN, where n

is odd, a processor $i, 0 \leq i < n$, is connected to processors $(i+1)\ mod\ n$, $(i-1)\ mod\ n$ and $2i\ mod\ n$. It was shown by Grosch that the multigrid algorithm can be implemented on a PSNN architecture with constant communication overheads for all the grids.

Chan and Schriber

Chan and Schriber [3] have considered Pyramid architecture for the implementation of the multigrid algorithm. A one dimensional pyramid of size k is defined as a set of k interconnected arrays of processors, one at each level. The one dimensional array at level $i, 0 \leq i < k$, contains 2^i processors, numbered from 0 to $2^i - 1$, connected linearly. The arrays of processors at different levels are connected such that processor j at level i is connected to processors $2j$ and $(2j + 1)$ at level $(i + 1)$. The same architecture can be extended to higher dimensions. Figure 1.5 shows a one dimensional pyramid of size four. The pyramid architecture is ideally suited for the implementation of the multigrid algorithm. Each grid of the multigrid hierarchy can be mapped onto one level of the pyramid. The computation on a grid of the multigrid hierarchy and operations involving data transfer among adjacent grids can be implemented with minimum communication overheads on the pyramid architecture. The major problem with this architecture is that except the grid on which processing is being done, the processors on all other grids remain idle. Authors have suggested that many multigrid problems must be processed in a pipeline fashion to fully utilize all the processors.

Kolp and Miernedorff

Implementation of the multigrid algorithm on meshes and tree architectures is discussed by Kolp and Miernedorff [22]. They consider the implementation of a d-dimensional problem onto a b-dimensional mesh, where b and d may be the same or different. The implementation is straightforward in the case where the dimension of the mesh is the same as the dimension of the problem. When the dimensions of the problem and that of the mesh do not match they give a more complicated implementation. Their implementation of the multigrid algorithm on mesh architecture does not solve the problem of high communication cost on very coarse grids.

The authors also consider the tree structured topology. For this topology, they map all the computational work onto the leaves of a tree such that the nearest neighbor communication is minimized. Since all the computation is done on the leaves of the tree, many of the processors are not utilized. They also show that the root of the tree becomes a bottleneck for the communication, because most of the traffic has to go through the root in their algorithm.

Chan and Saad

Chan and Saad [4] have suggested the hypercube architecture for implementation of the multigrid algorithm. An n-dimensional hypercube has 2^n processors, numbered from 0 to $2^n - 1$. Processor i is connected to processor j if the binary representation of i and j differ in exactly one bit. A three dimensional hypercube is shown in Figure 1.6. Chan and Saad have shown that by using binary reflected gray code

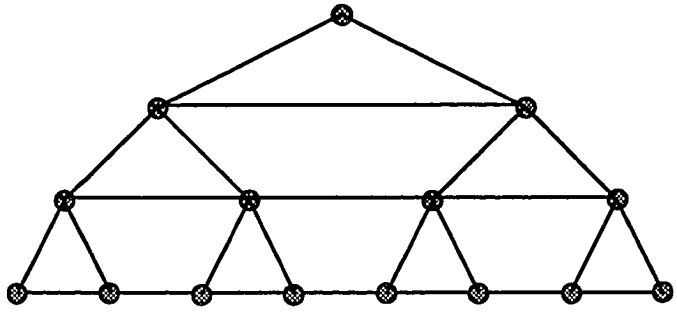


Figure 1.5 : A one dimensional pyramid of size four

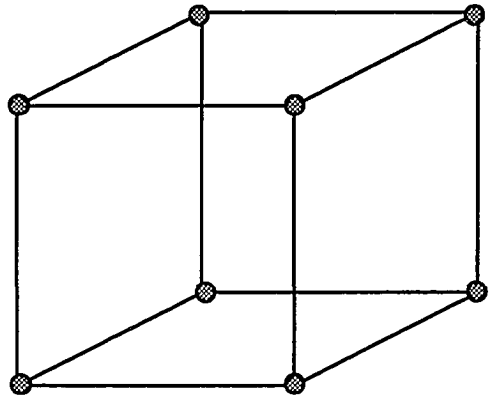


Figure 1.6 : A three dimensional hypercube

mapping, the multigrid hierarchy can be mapped onto a hypercube without much communication overheads. In their implementation, the communication on the finest grid requires communication between adjacent processors only. For coarser grids, the distance between two processors which need to communicate is at the most two.

Although the architectures discussed above have many interesting features for the implementation of multigrid algorithms, in practice very large commercially available systems have been mostly built using either two dimensional mesh architecture (AMT-DAP, MASPAR, Touchstone Delta, Paragon) or hypercube architecture (Connection machine, Intel iPSC/860, N-Cube). The present trend in the very large parallel system architectures is mesh connected systems, because of its easy scalability.

1.2.2 Modified Multigrid Algorithms

There are several approaches proposed in the literature which modify the multigrid algorithm in order to improve the processor utilization on massively parallel computers. Most of these algorithms fall into the category of the first approach of our classification. That is to do extra computational work in a multigrid iteration and to try to accommodate this extra work on the unutilized processors. There is no known work based on the second approach.

Gannon and Rosendale

Gannon and Rosendale [15] were among the first to present an alternative multigrid algorithm called the Concurrent Iteration Multigrid (*CIMG*) algorithm. In the *CIMG* algorithm, computation on all the grid levels is done simultaneously as opposed to the standard multigrid algorithm where the computation is done only on one grid level at a given time. The basic idea of their algorithm is to distribute current residuals on the finest grid to all the grid levels. This is followed by concurrent relaxation iterations on all the grids. In the final step, results from all the grids are combined to obtain a new approximate solution on the finest grid. The *CIMG* algorithm achieves a better performance than the standard multigrid algorithm for several parallel architectures. One of the problems with their algorithm is that it requires more number of processors than the standard multigrid algorithm because all the grids are processed simultaneously. Further, the performance of their algorithm is highly dependent on the communication topology of the parallel machine.

Greenbaum

A similar idea to the *CIMG* algorithm is used by Greenbaum [17] and also by Swisshelm et al [30]. In their method, before doing any relaxation on the fine grid, the residuals are projected to each of the coarse grids. Then on each grid a system of linear equations is solved approximately using the projected residuals as the right hand side. In the algorithm by Greenbaum, a method is proposed for collecting solutions from each of the grids, so that in the combined solution A-norm

of the error is minimized. For the parallel machines their algorithm also has similar advantages and disadvantages as those of the *CIMG* algorithm.

Fredrickson and Mcbryan

Recently Fredrickson and Mcbryan [12, 7, 13] have suggested a Parallel Superconvergent Multigrid (*PSMG*) algorithm. The central idea in the *PSMG* algorithm is to use multiple grids on the coarse levels. For example, in a one-dimensional problem, there can be two coarse grids, one corresponding to the even numbered fine grid points and the second corresponding to the odd numbered fine grid points. Since in the standard multigrid algorithm only half the processors are used on the coarse grid, the remaining half can work on the second coarse grid. The solutions from the two coarse grids are combined while interpolating them on the fine grid to obtain a better convergence rate. The *PSMG* algorithm obtains significant improvement in convergence over the standard multigrid method for the solution of the Poisson equation. The performance of the *PSMG* algorithm is highly dependent on the choice of prolongation and restriction operators. If the chosen operators are not appropriate then one can get a slower convergence or even divergence. An evaluation of the parallel performance of the *PSMG* algorithm is given by Decker [8, 9].

Chan and Tuminaro

The algorithm proposed by Chan and Tuminaro [6, 31] for handling the problem of low processor utilization is similar in principle to the *CIMG* algorithm of Gannon and Rosendale. In their algorithm, the residual on the fine grid is split into two

components, one containing low frequency components and the other containing middle and high frequency components. The low frequency component is projected onto the coarse grid, and the other component is kept on the fine grid itself. The splitting is followed by concurrent relaxation on both grids to obtain a superior convergence rate. The problems with this algorithm are similar to those with the *CIMG* algorithm, and also it is relatively difficult to program. In addition, when the overheads in splitting are taken into account, the improvement in convergence rate is not significant.

1.3 Our Approach

In this dissertation, we propose modified multigrid algorithms which address the problem of communication cost and processor utilization. The proposed algorithms follow the two approaches described in our classification. The first algorithm, the *Overlap Parallel Multigrid (OPMG)* algorithm, increases the work done during a multigrid cycle and tries to accommodate this increased work on the unutilized processors. The increased work is generated by modifying the multigrid algorithm such that while computation is being done on a coarse grid, extra computation is done on the next finer grid also. In the *OPMG* algorithm, this extra computation, which was not present in the standard multigrid algorithm, is performed concurrently with the coarse grid computation. In this dissertation, we have shown that the extra fine grid computation improves the convergence rate of the multigrid algorithm. However, for the *OPMG* algorithm to be useful, it is necessary that there is not much

overhead in overlapping the extra fine grid computation with the coarse grid computation. This is a challenging problem for SIMD machines, where any irregularity in the computation implies additional overheads. It gets even worse for machines like the AMT-DAP which do not support indirect addressing. To solve this problem, we have developed an implementation strategy which allows concurrent computation on the coarse and fine grids for SIMD machines. The implementation of our strategy on the AMT-DAP causes very minimal overheads.

The second algorithm we propose is the *Chopped Parallel Multigrid (CPMG)* algorithm. The *CPMG* algorithm improves the processor utilization by reducing the work load on coarse grids without affecting the convergence rate of the algorithm. This is in contrast to the first approach, where unutilized processors are used to improve the convergence rate. The *CPMG* algorithm reduces the coarse grid work by *chopping* alternate cycles of the multigrid algorithm. Using analytical results and simulations on sequential machines, we show that the *CPMG* algorithm can achieve almost the same convergence rate as the standard multigrid algorithm in many cases. On the other hand, the *CPMG* algorithm gives significant advantage in terms of average parallel execution time per cycle due to the reduction of work on the coarse grids.

Finally, we give a hybrid algorithm, which combines the advantages of the *OPMG* and the *CPMG* algorithms. Since both these algorithms are based on complementary philosophies, they can be effectively combined into the hybrid algorithm, which gives better results than either the *OPMG* algorithm or the *CPMG* algorithm individually.

1.4 Thesis Outline

The rest of the dissertation is organized as follows. In Chapter 2, we discuss background material which will be used in the following chapters. This includes a detailed description of the multigrid algorithm, a description of the parallel machine AMT-DAP, which is used for implementation of our algorithms and a description of problems used for testing our algorithms. Chapter 3 contains a description of the *OPMG* algorithm. The convergence of the *OPMG* algorithm and its performance on parallel machines are also discussed in this chapter. The *CPMG* algorithm is the subject matter of Chapter 4. In Chapter 5, we give a description and the results for the hybrid algorithm, which is a combination of the two algorithms discussed in Chapter 3 and Chapter 4. Finally in Chapter 6, we give conclusions of our research work and a few suggestions about how this work can be extended.

Chapter 2
Background

2.1 Multigrid Algorithm

The multigrid algorithm is one of the most popular numerical methods for solving partial differential equations. It is mostly used for solving *elliptic pde*'s but has been used for other types of *pde*'s also [20, 21, 27]. Approximate numerical solution of a *pde* is usually obtained by discretizing the problem domain using a grid. On each grid point of the domain, the partial differential operator is transformed into a finite difference operator using a finite difference scheme such as the *central difference* scheme. A detailed discussion on finite difference schemes can be found in [25]. The discretization process transforms the partial differential equation on each grid point into a linear equation. These linear equations on all the grid points form a system of linear equations. Schemes to solve such systems of linear equations can be classified into two categories, direct schemes such as Gaussian elimination

and relaxation (iterative) schemes such as Jacobi, Gauss-Seidel, SOR, etc. The relaxation schemes start with an initial approximate solution and iteratively improve it, until it converges to the exact solution within a desired level of accuracy. The problem with relaxation schemes is that their convergence slows down significantly after a few initial iterations. A typical behavior of the weighted Jacobi scheme in terms of convergence is shown in Figure 2.1. Here, we have plotted the number of iterations on the X-axis and the $\|.\|_2$ norm of error the (difference between the approximate solution and the exact solution) on the Y-axis. The cause for the slow convergence rate is the property that these schemes are good for removing oscillatory (high frequency) components of the error, but perform poorly for smooth (low frequency) components of the error [19]. Here, the frequency corresponds to spatial domain rather than time domain. Due to this property, the convergence rate of these schemes is good during the first few iterations, which are responsible for removing the high frequency components of the error. The convergence rate drops down significantly afterwards because the leftover low frequency error components cannot be removed effectively.

The multigrid algorithm solves this problem by using a hierarchy of grids with increasing grid spacing, in addition to the grid on which the solution is sought. Usually a coarse grid is formed by doubling the grid spacing of the next finer grid. In other words, the grid points of a coarse grid correspond to alternate grid points of the next finer grid. The central idea of the multigrid algorithm is that once the high frequency components of the error are removed on a grid using a relaxation

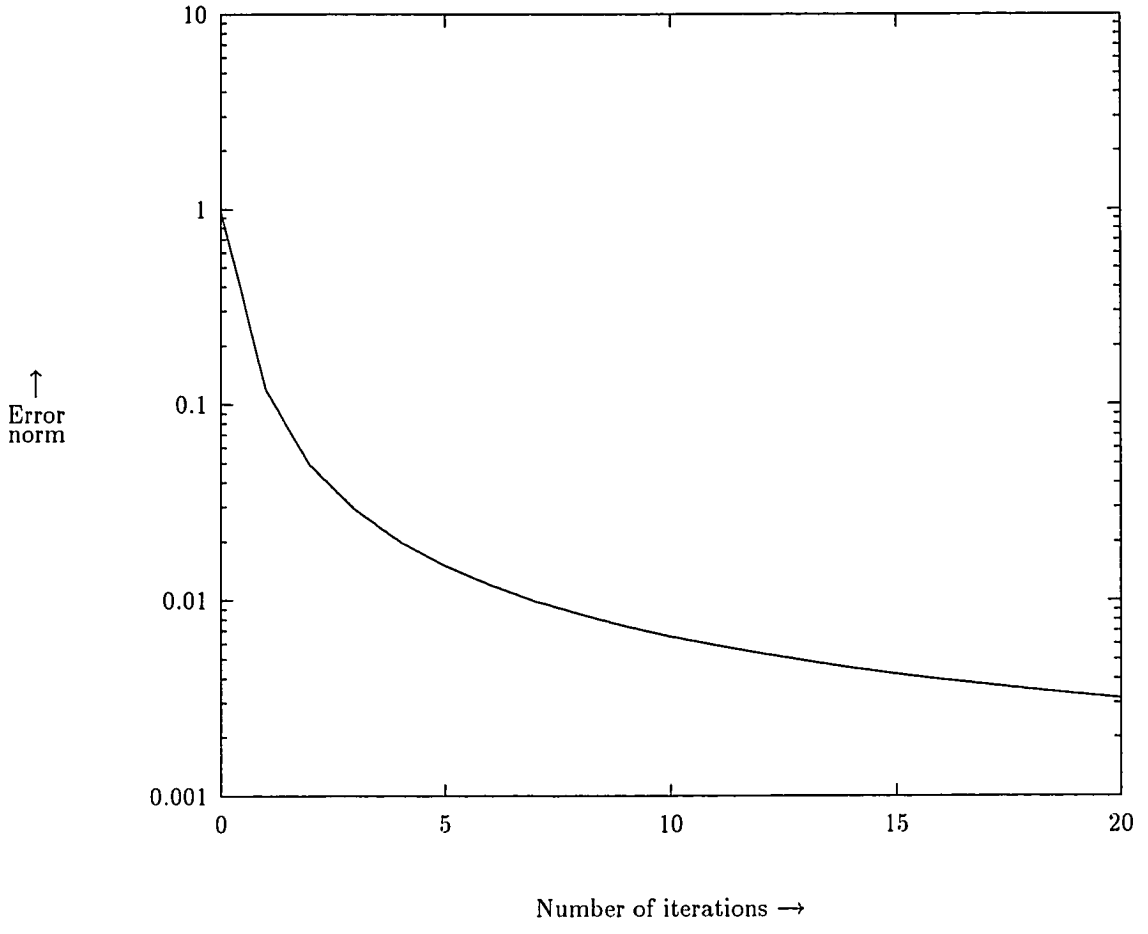


Figure 2.1: Convergence behavior of Weighted Jacobi relaxation scheme

scheme, the problem is transferred to a coarser grid. On this grid the low frequency components of the error become relatively high frequency components with respect to the grid spacing. Thus, these error components can be removed effectively on the coarse grid using relaxation schemes. The recursive application of this idea gives the multigrid algorithm.

For a precise description of the multigrid algorithm we use the following notations.

- $G^i, 1 \leq i \leq n$ represents the ith grid in a hierarchy of grids, where G^1 is the coarsest grid and G^n is the finest grid.
- L^i represents discretization of the differential operator on G^i.
- v^i ,f^i and r^i represent the approximate solution, the forcing function and the residual on G^i respectively.
- P^{i-1} represents the projection operator, which projects values from G^i to G^{i-1}. This operator is also referred to as the restriction operator.
- I^i represents the interpolation operator, which interpolates values from G^{i-1} to G^i. This operator is also referred to as the prolongation operator.

With the above notation the multigrid algorithm can be described as :

Algorithm **MG** $(L^i, f^i, v^i, i, s, coarsest)$

begin

if (G^i is *coarsest* grid) **then**

if *coarsest* is 1 **then** Solve $L^1 v^1 = f^1$

else Perform s iterations of relaxation on G^i

endif

else

Perform s iterations of relaxation on G^i (Pre-relax)

Compute residual on G^i : $r^i = f^i - L^i v^i$.

Project residual to G^{i-1} : $f^{i-1} = P^{i-1} r^i$

Initialize solution on G^{i-1} : $v^{i-1} = 0$

Solve on G^{i-1}, $L^{i-1} v^{i-1} = f^{i-1}$:

Call $\mathbf{MG}(L^{i-1}, f^{i-1}, v^{i-1}, i-1, s, coarsest)$

Interpolate correction to G^i : $v^i = v^i + I^i v^{i-1}$.

Perform s iterations of relaxation on G^i (Post-relax)

endif

end

The above multigrid algorithm is usually referred to as the V-cycle multigrid algorithm. The coarsest grid is G^1 for the V-cycle multigrid algorithm. In Figure 2.2 we give a pictorial representation of the V-cycle multigrid algorithm for a four grid hierarchy and in Figure 2.3 we give the sequence of operations performed in one V-cycle.

There are many variants of the multigrid algorithm such as W-cycle multigrid algorithm, Full Multigrid algorithm, Full Approximation Storage multigrid algorithm etc [1]. In this dissertation we will concentrate on the V-cycle multigrid algorithm

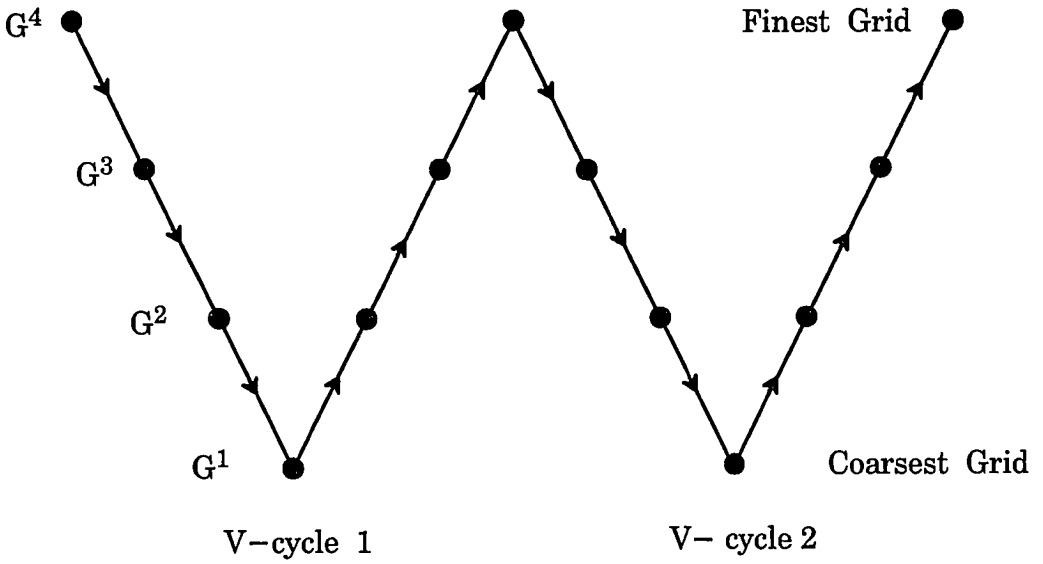


Figure 2.2 : Pictorial representation of V−cycle multigrid algorithm

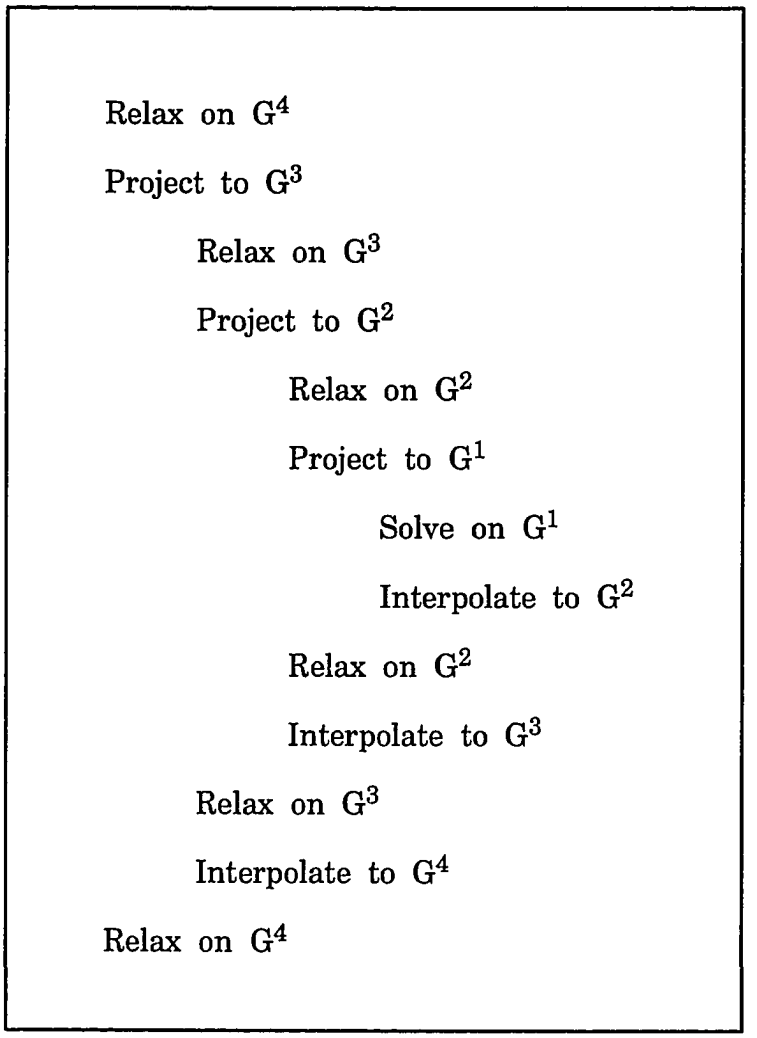


Figure 2.3 : Sequence of operations for one V–cycle

because it is the most popular variant of the multigrid algorithm and is suitable for parallel implementation.

2.2 Architecture of AMT-DAP/510

The performance of the algorithms proposed in this dissertation is evaluated on a fine grain massively parallel computer AMT-DAP/510 [26]. It is a SIMD machine with 1024 one-bit processors. The processors of the DAP are arranged in a 32×32 matrix and each processor is connected to its four nearest neighbors. The processors on the edge of the matrix have wrap around connections to the processors on the opposite edge (Figure 2.4). In addition to the nearest neighbor connections, a bus system connects all the processors in each row and all the processors in each column. These buses are referred to as the *row bus* and the *column bus* respectively. Each processor has a local memory of 64K bits. The whole memory can be viewed as a three dimensional array of bits, consisting of 64K *bit-planes*. A bit-plane has 1024 bits, one from each processor's local memory at the same address (Figure 2.5). Similarly, a *word-plane* has 1024 words, one from each processor's local memory.

In the DAP, the processor array is controlled by a Master Control Unit (MCU), which acts as a source of instructions for the processor array (Figure 2.6). The object code of a DAP program is loaded into the MCU's memory, from which the MCU fetches instructions and interprets them. Some of the fetched instructions are executed in the MCU itself (scalar instructions, control instructions, etc.), whereas others are broadcast to the processor array. Since the same instruction is broadcast

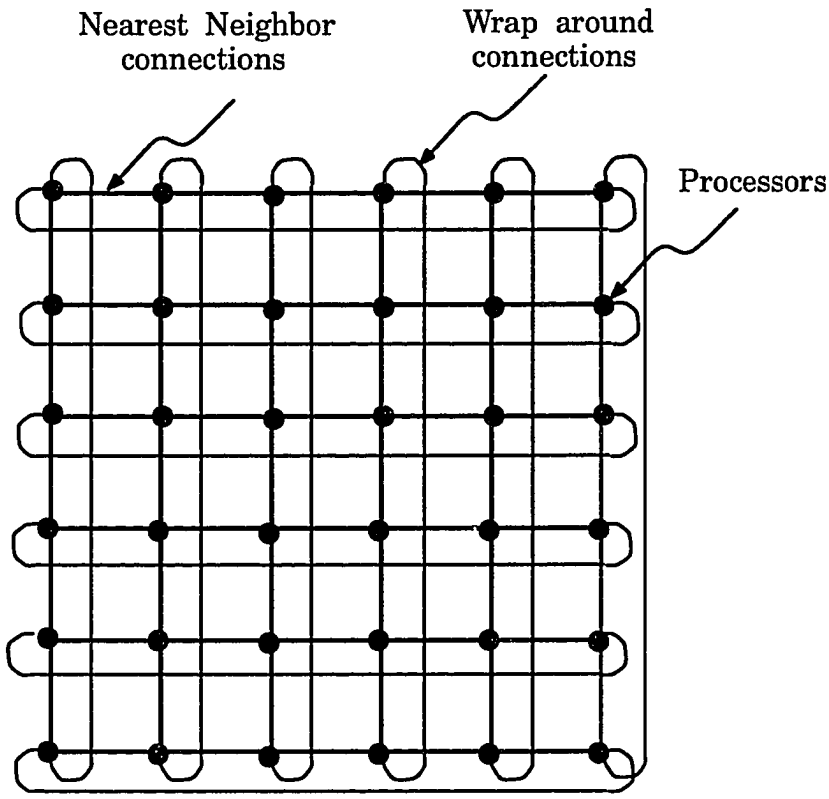


Figure 2.4 : Processor interconnection topology for AMT−DAP

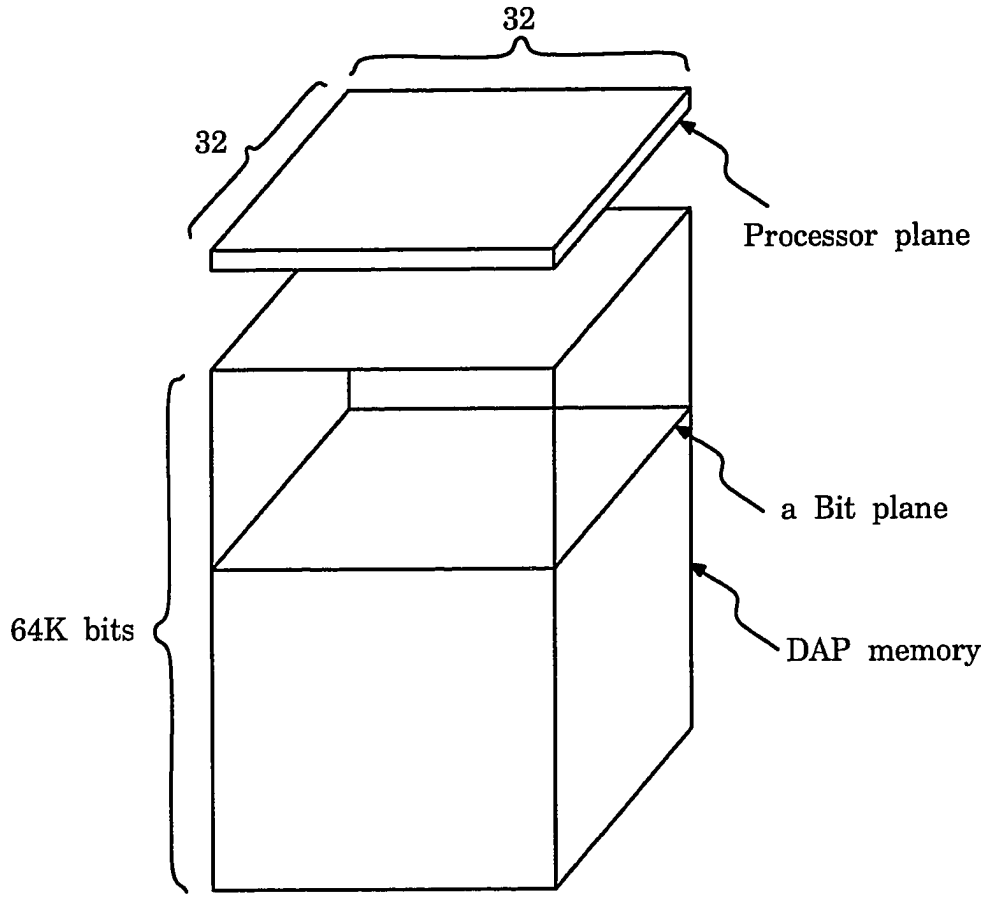


Figure 2.5 : Memory architecture of AMT–DAP

to all the processors in the processor array, all of them perform the same operation. The operand addresses are also broadcast by the MCU, which implies that all the processors act on the data at the same address in their local memory. This mode of execution puts a restriction on indirect addressing, where each processor can operate on data at a different address in its local memory.

The high level language available on the DAP is an extension of FORTRAN-77, called FORTRAN-PLUS [14]. The most important feature of FORTRAN-PLUS is its ability to manipulate matrices and vectors. For example, two matrices can be added with a single statement, as is done for scalars. The masking and selection operations are available for performing computation on selected processors. Now we briefly describe some of the operations used for implementation of the algorithms proposed in this dissertation on DAP.

Computation: $A = B + C$

This operation adds two matrices B and C and stores the result in matrix A. Similar operation can also be performed on vectors.

Communication: $A = shep(B, len)$

The *shep* operation shifts the matrix B in the *East* direction by an amount *len*. The results are assigned to another matrix A.

Masking: $A(mask) = B$

This operation assigns values of matrix B to matrix A, wherever the value of the logical matrix *mask* is true. The remaining values of matrix A are not changed.

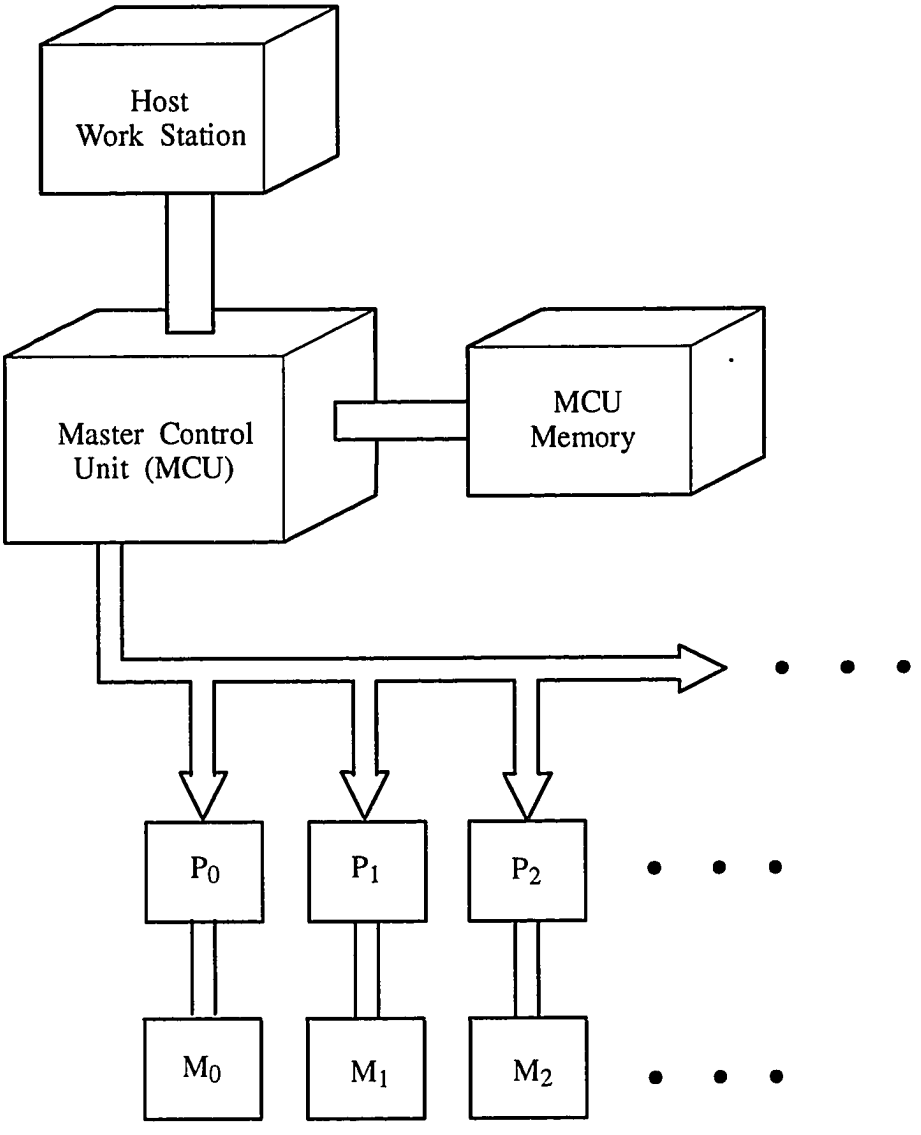


Figure 2.6 : Master Control Unit (MCU) of AMT–DAP

2.3 Test Problems

2.3.1 Description of Test Problems

The algorithms presented in this dissertation are tested using a number of problems. These problems require solution of the Poisson equation over a two dimensional domain with zero boundary conditions. The following equation and solution functions describe all the three test problems used.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad 0 < x < 1, \quad 0 < y < 1$$

$$u(0,y) = u(1,y) = u(x,0) = u(x,1) = 0$$

Problem 1 :

$$u(x,y) = random$$

The solution contains all the frequency components with random magnitudes.

Problem 2 :

$$u(x,y) = sin(\pi x) sin(\pi y)$$

The solution contains only lowest frequency components.

Problem 3 :

$$u(x,y) = \sum_{k=1}^{N}\sum_{l=1}^{N} \frac{sin(k\pi x)sin(l\pi y)}{N+1},$$

N = *number of grid points in each dimension*

The solution contains all the frequency components with equal magnitudes.

For testing our algorithms using these problems, we assume the initial approximation to the required solution to be zero on all the grid points. This implies that the initial error in the approximate solution is equal to the required solution. The known nature of the initial error helps in analyzing the behavior of our algorithms for these problems. In terms of the nature of the initial error, these problems represent a wide variety of situations which occur while solving physical problems. The initial error for Problem 1 is random, which implies the error contains all the frequency modes with random magnitudes. For problem 2, the initial error contains only the lowest frequency component. This problem represents the worst case because very low frequency components of error are most difficult to remove using relaxation methods and also multigrid algorithm. The initial error for Problem 3 contains all the frequency modes with equal magnitudes.

2.3.2 Simulation of Test Problems

For simulation of each problem we first compute the function $f(x,y)$ at each grid point by applying the discrete form of the Laplacian to the known solution $u(x,y)$.

Once $f(x, y)$ is computed, we assume that the exact solution $u(x, y)$ is not known and use the algorithm being tested to find an approximation to the exact solution $u(x, y)$. The initial approximation to the solution is assumed to be zero in all the cases. We use weighted Jacobi relaxation scheme [32] in all the simulations.

Chapter 3
Overlap Parallel Multigrid Algorithm

3.1 Introduction

In this chapter, we present the *Overlap Parallel Multigrid (OPMG)* algorithm for massively parallel SIMD machines. The *OPMG* algorithm follows the first approach of our classification given in Chapter 1 (Figure 1.4), for addressing the problem of low processor utilization. This approach increases the work in each V-cycle to improve the convergence rate of the multigrid algorithm and tries to accommodate the increased work on the unutilized processors of the parallel machine.

In the *OPMG* algorithm, the work done during a V-cycle is increased by doing extra iterations of relaxation on each grid of the multigrid hierarchy. These extra iterations of relaxation are organized such that they can be done concurrently with the relaxation iterations on the next coarse grid. To differentiate this extra computation from normal computation done in a V-cycle, we refer to it as extra fine grid

computation or extra fine grid iteration. A similar algorithm for data driven multiprocessors was proposed in [23]. For a better understanding of the computation in the *OPMG* algorithm, consider a multigrid hierarchy of four grids as shown in Figure 3.1. In the V-cycle multigrid algorithm, $s1$ iterations of relaxation are done on grid G^4 and then the problem is projected on the next coarse grid G^3. The same process is repeated for the grids G^3, G^2 and G^1 (Figure 2.3). On the other hand, in the *OPMG* algorithm extra $s2$ iterations of relaxation are done on grid G^4 *after* the problem has been projected to grid G^3. These extra iterations of relaxation are done before the corrections from grid G^3 are interpolated to grid G^4. The same process is recursively repeated for the rest of the grids. Figure 3.2 gives the sequence of operations performed by the *OPMG* algorithm for the four grid hierarchy shown in Figure 3.1.

The increased work in the *OPMG* algorithm is useful only if (i) the convergence rate for the *OPMG* algorithm improves in comparison to the standard V-cycle multigrid algorithm, and (ii) the increased work can be accommodated on unutilized processors without much overheads. In the later sections of this chapter we will present analytical and simulation results to show that the *OPMG* algorithm does indeed obtain a better convergence rate because of the extra fine grid computation. The improvement in convergence rate can be intuitively explained by observing that the extra fine grid computation moves the present approximate solution on a grid closer to the exact solution by removing high frequency components of the error. But, one may argue that this advantage can be offset because the corrections from

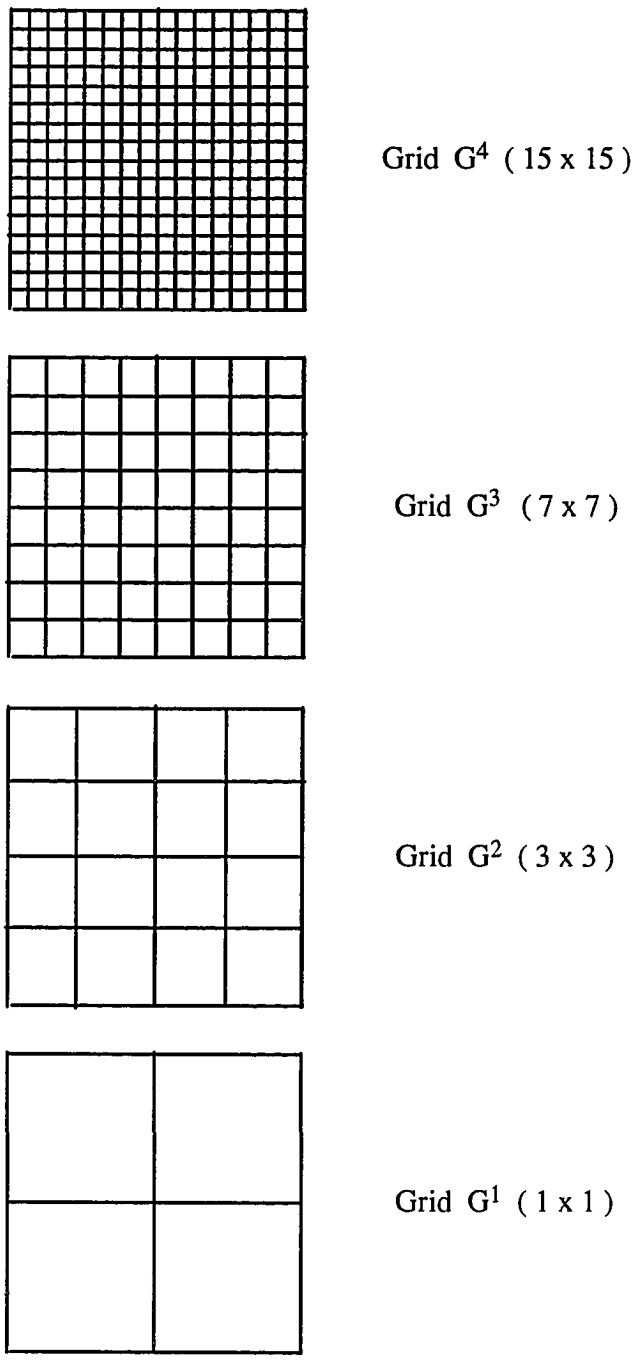


Figure 3.1 : A multigrid hierarchy of four grids

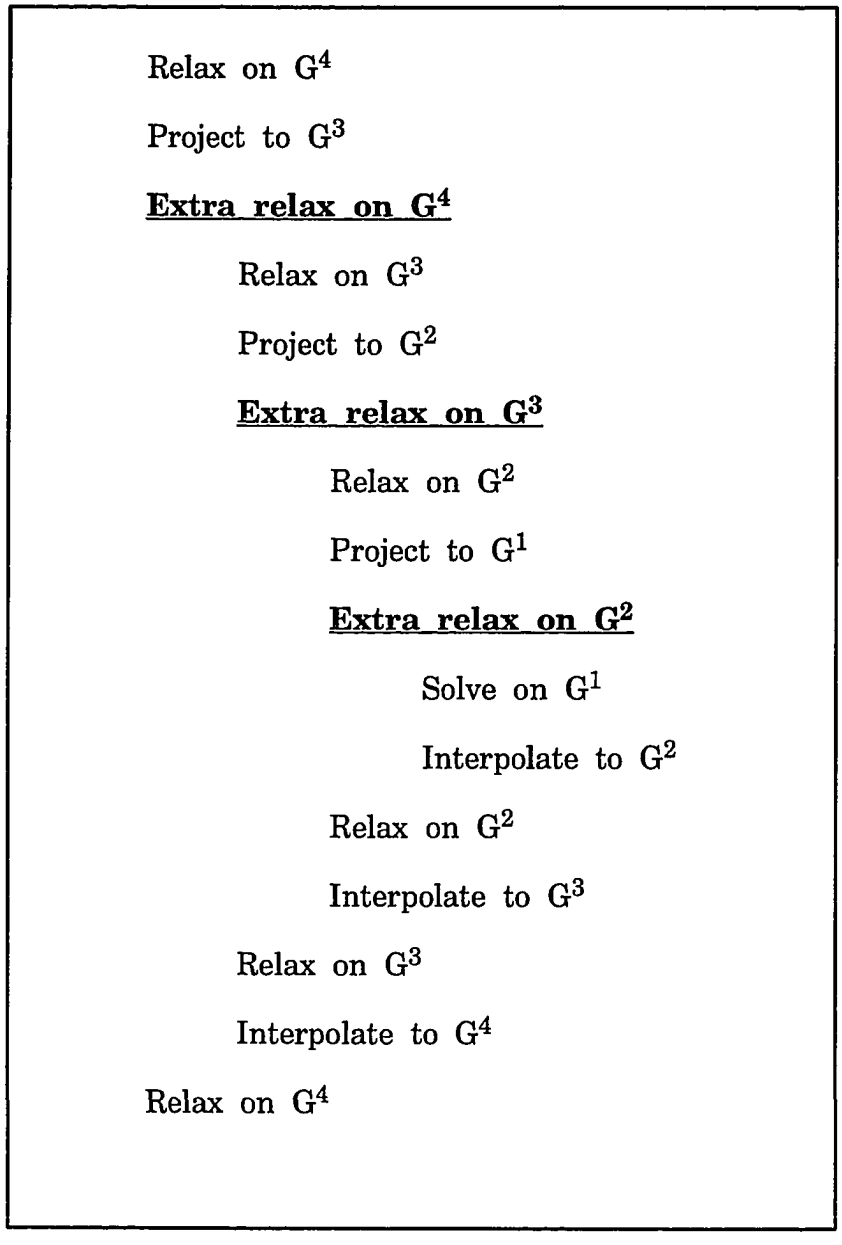


Figure 3.2 : Sequence of operations for a four grid *OPMG*

the next coarse grid are intended for the approximate solution before the extra fine grid computation is done. In the *OPMG* algorithm they are applied to the approximate solution after the extra fine grid computation is done. However, this is not a problem because the corrections from the next coarse grid pertain to low frequency components of the error on which the extra fine grid computation has very little impact. For ensuring the second point that the extra fine grid computation in the *OPMG* algorithm can be done without significant increase in parallel execution time, we make use of the unutilized processors during computation on the coarse grids. As explained earlier, the extra fine grid computation is done *after* projecting the problem to the next coarse grid. This implies that the extra fine grid computation on a grid and the normal computation on the next coarse grid can be done concurrently. Since the normal computation on the next coarse grid does not use all the processors, the unutilized processors can be used for doing extra fine grid computation. This is a challenging problem on SIMD machines because any irregularity in computation gives rise to additional computation cost. In this case, the irregularity arises because some of the processors need to work on the coarse grid points and the others on the fine grid points. We overcome this problem by using a complex masking and communication scheme and a hybrid data structure, details of which are given in later sections.

Now we give a concise description of the *OPMG* algorithm. We use the same notations as for the standard V-cycle multigrid algorithm described in Chapter 2. For the *OPMG* algorithm we have an additional parameter $s2$ which gives the number

of extra iterations of relaxation.

Algorithm **OPMG** $(L^i, f^i, v^i, i, s1, s2, coarsest)$

begin

if (G^i is *coarsest* grid) **then**

if *coarsest* is 1

then Solve $L^1 v^1 = f^1$

else Perform s iterations of relaxation on G^i

endif

else

Perform $s1$ iterations of relaxation on G^i (Pre-relax)

Compute residual on G^i : $r^i = f^i - L^i v^i$.

Project residual to G^{i-1} : $f^{i-1} = P^{i-1} r^i$

Perform $s2$ extra iterations of relaxation on G^i

Initialize solution on G^{i-1} : $v^{i-1} = 0$

Solve on G^{i-1}, $L^{i-1} v^{i-1} = f^{i-1}$:

Call **OPMG**$(L^{i-1}, f^{i-1}, v^{i-1}, i-1, s1, s2, coarsest)$

Interpolate correction to G^i : $v^i = v^i + I^i v^{i-1}$.

Perform $s1$ iterations of relaxation on G^i (Post-relax)

endif

end

In the above description, the extra relaxation step appears before the computation on the next coarse grid. However, it can be done any time after the problem has

been projected to the next coarse grid and before the corrections from the next coarse grid are applied to the approximate solution.

3.2 Convergence Analysis of *OPMG* Algorithm

Convergence analysis of the V-cycle multigrid algorithm is usually done with a two grid model which closely approximates the behavior of the V-cycle multigrid algorithm [19]. The two grid model consists of a fine grid and a coarse grid . In this model, it is assumed that the exact solution of the coarse grid problem can be computed. We use this model for comparing the convergence rate of the *OPMG* algorithm with that of the standard V-cycle multigrid algorithm.

The following is a brief description of the two-grid *OPMG* algorithm. Here, we use G^n for the fine grid and $G^{(n-1)}$ for the coarse grid. The effect of s iterations of relaxation on the approximate solution is given in terms of matrices S^n and R_s^n, which define the relaxation method[16, 19].

Algorithm **Two-grid OPMG**

begin

Relax $s1$ times on G^n:	$v^n = (S^n)^{s1} v^n + R_{s1}^n f^n$
Compute residual on G^n:	$r^n = (f^n - L^n v^n)$
Project residual on G^{n-1}:	$f^{n-1} = P^{n-1} r^n$
Relax $s2$ times on G^n:	$v^n = (S^n)^{s2} v^n + R_{s2}^n f^n$
Get solution on G^{n-1}:	$v^{n-1} = (L^{n-1})^{-1} f^{n-1}$

Interpolate solution to G^n: $v^n = v^n + I^n v^{n-1}$

Relax $s1$ times on G^n: $v^n = (S^n)^{s1} v^n + R^n_{s1} f^n$

end

The net effect of the two grid *OPMG* algorithm on the approximate fine grid solution v^n can be computed by combining the effects of the operations mentioned above. In the following, we use S in place of S^n and R_s in place of R^n_s, for simplicity.

$$\begin{aligned} v^n &= S^{s1}(S^{s1+s2} v^n + R_{s1} f^n + R_{s2} f^n + \\ &\quad I^n (L^{n-1})^{-1} P^{n-1} (f^n - L^n (S^{s1} v^n + R_{s1} f^n)) + R_{s1} f^n \end{aligned} \tag{1}$$

Assuming that u^n is the exact solution, the above equation should also hold for u^n.

$$\begin{aligned} u^n &= S^{s1}(S^{s1+s2} u^n + R_{s1} f^n + R_{s2} f^n + \\ &\quad I^n (L^{n-1})^{-1} P^{n-1} (f^n - L^n (S^{s1} u^n + R_{s1} f^n)) + R_{s1} f^n \end{aligned} \tag{2}$$

Subtracting Equation 1 from Equation 2, we get

$$(u^n - v^n) = S^{s1}(S^{s1+s2}(u^n - v^n) - I^n (L^{n-1})^{-1} P^{n-1} (L^n S^{s1} (u^n - v^n)))$$

The above equation can be written in terms of the error in the approximate solution ($e^n = u^n - v^n$).

$$e^n = (S^{s1}(S^{s1+s2} - I^n (L^{n-1})^{-1} P^{n-1} (L^n S^{s1})))e^n \tag{3}$$

Equation 3 gives the iteration matrix for the two grid *OPMG* algorithm. Iteration matrix for the V-cycle multigrid algorithm can be obtained by making $s2$ equal to

zero in the above equations, which implies that no extra iteration of relaxation are done on the fine grid. For comparing the two algorithms, we compute the spectral radius of the iteration matrix for the following one dimensional model problem [19].

$$u''(x) = f, \qquad 0 < x < 1$$

$$u(0) = u(1) = 0$$

The finite difference discretization of this problem using a one dimensional grid of $(N+1)$ points results in a system of linear equations, where $N = 2^i$.

$$A\mathbf{u} = \mathbf{f}$$

The matrix A has a tridiagonal structure with all of the diagonal elements equal to 2 and all the super and sub-diagonal elements equal to -1.
We use the following operators for our analysis.

- Relaxation (S^i) : Weighted Jacobi method[32] with weight equal to half.
- Projection (P^i) : Full weighting projection operator[2].
- Interpolation (I^i) : Bilinear interpolation operator[2].

The following table gives the spectral radius for the *OPMG* algorithm with various numbers of extra iterations. We have followed the approach suggested in [19] for computing the spectral radius. All the results in this table are computed with the value of $s1$ equal to one.

Table 3.1 : Spectral radius for the *OPMG* algorithm

Extra Iterations($s2$)	Spectral Radius
0(MG)	$\frac{1}{4}$
1	$\frac{1}{8}$
2	$\frac{3}{16}$

The results in this table indicate that the spectral radius for the *OPMG* algorithm is smaller than that of the V-cycle multigrid algorithm for the chosen model problem. Since the spectral radius of the iteration matrix gives the asymptotic convergence rate of the algorithm, it is clear that the *OPMG* algorithm achieves a better convergence rate than the V-cycle multigrid algorithm.

3.3 Simulation Results for *OPMG* Algorithm's Convergence

The convergence rates of the *OPMG* and the V-cycle multigrid algorithms were also compared by doing simulations on sequential machines. The simulated problems require the solution of the two dimensional Poisson equation with different forcing functions. The finest grid used for discretization of the Poisson equation has 511 points in each dimension excluding boundary points. We use the test problems described in Chapter 2 for our simulations.

The results of our simulations are presented in Table 3.2 to Table 3.4. Each

of these tables gives the convergence history of one problem for both the V-cycle multigrid algorithm and the *OPMG* algorithm. The convergence history is given in terms of the $\|.\|_2$ norm of the residual and the maximum absolute residual after each cycle of the two algorithms. In these tables, we have highlighted the entries which correspond to the same level of convergence. This gives the number of cycles required by the *OPMG* algorithm to reach the same level of convergence as that achieved by the V-cycle multigrid algorithm after 10 cycles.

The results in these tables indicate that the *OPMG* algorithm achieves a better convergence rate than the V-cycle multigrid algorithm for all the three problems. This can be seen from the fact that the residual norm after the last cycle of the *OPMG* algorithm is smaller than that of the V-cycle multigrid algorithm for all the test problems. By looking at these results from a different perspective, we see that, the *OPMG* algorithm reduces the residual norm to the same level as the V-cycle multigrid algorithm in much less number of iterations. This is shown in the tables by highlighting the corresponding entries.

We have also depicted the results of our simulations in graphs of Figure 3.3 to Figure 3.5. In all the graphs, the number of cycles is plotted on the X-axis and the normalized $\|.\|_2$ norm of the residual is plotted on the Y-axis. The normalized residual norm is computed by taking the ratio of the current residual norm to the initial residual norm. These results also show that the *OPMG* algorithm achieves a better convergence rate in comparison to the V-cycle multigrid algorithm.

Finally, we summarize the results of our simulations in Table 3.5 for all the

Table 3.2 : Convergence history comparison for Problem 1

No. of cycles	V-cycle residual norm	*OPMG* residual norm	V-cycle max residual	*OPMG* max residual
0	1.28E+0	1.28E+0	3.76E+0	3.76E+0
1	1.22E-1	6.24E-2	5.91E-1	3.76E-1
2	3.40E-2	1.36E-2	2.22E-1	9.19E-2
3	1.25E-2	3.29E-3	8.92E-2	2.48E-2
4	4.90E-3	8.68E-4	3.69E-2	6.75E-3
5	1.98E-3	2.33E-4	1.53E-2	1.88E-3
6	8.16E-4	6.42E-5	6.48E-3	5.30E-4
7	3.41E-4	**1.78E-5**	2.78E-3	1.50E-4
8	1.44E-4	5.00E-6	1.20E-3	4.26E-5
9	6.10E-5	1.41E-6	5.16E-4	1.23E-5
10	**2.60E-5**	4.22E-7	2.23E-4	3.73E-6

Table 3.3 : Convergence history comparison for Problem 2

No. of cycles	V-cycle residual norm	*OPMG* residual norm	V-cycle max residual	*OPMG* max residual
0	3.76E-5	3.76E-5	7.53E-5	7.53E-5
1	2.01E-5	1.99E-5	3.45E-4	4.21E-4
2	8.46E-6	7.96E-6	5.45E-5	2.48E-4
3	3.67E-6	1.19E-6	2.99E-5	5.48E-5
4	1.57E-6	4.37E-7	9.36E-6	9.95E-6
5	6.74E-7	1.60E-7	4.65E-6	2.38E-6
6	3.06E-7	1.20E-7	2.15E-6	8.34E-7
7	1.67E-7	**1.18E-7**	1.19E-6	8.34E-7
8	1.28E-7	1.18E-7	9.54E-7	7.15E-7
9	1.19E-7	1.18E-7	8.34E-7	7.15E-7
10	**1.18E-7**	1.18E-7	7.75E-7	7.15E-7

Table 3.4 : Convergence history comparison for Problem 3

No. of cycles	V-cycle residual norm	*OPMG* residual norm	V-cycle max residual	*OPMG* max residual
0	1.73E-2	1.73E-2	6.46E+0	6.46E+0
1	2.02E-3	1.10E-3	7.04E-1	3.36E-1
2	6.27E-4	2.50E-4	1.79E-1	6.13E-2
3	2.32E-4	6.19E-5	5.58E-2	1.21E-2
4	9.10E-5	1.61E-5	1.87E-2	2.57E-3
5	3.66E-5	4.32E-6	6.52E-3	5.80E-4
6	1.50E-5	1.17E-6	2.34E-3	1.51E-4
7	6.21E-6	**3.23E-7**	8.61E-4	4.07E-5
8	2.60E-6	8.95E-8	3.37E-4	1.09E-5
9	1.09E-6	2.49E-8	1.39E-4	2.91E-6
10	**4.64E-7**	7.08E-9	5.86E-5	8.25E-7

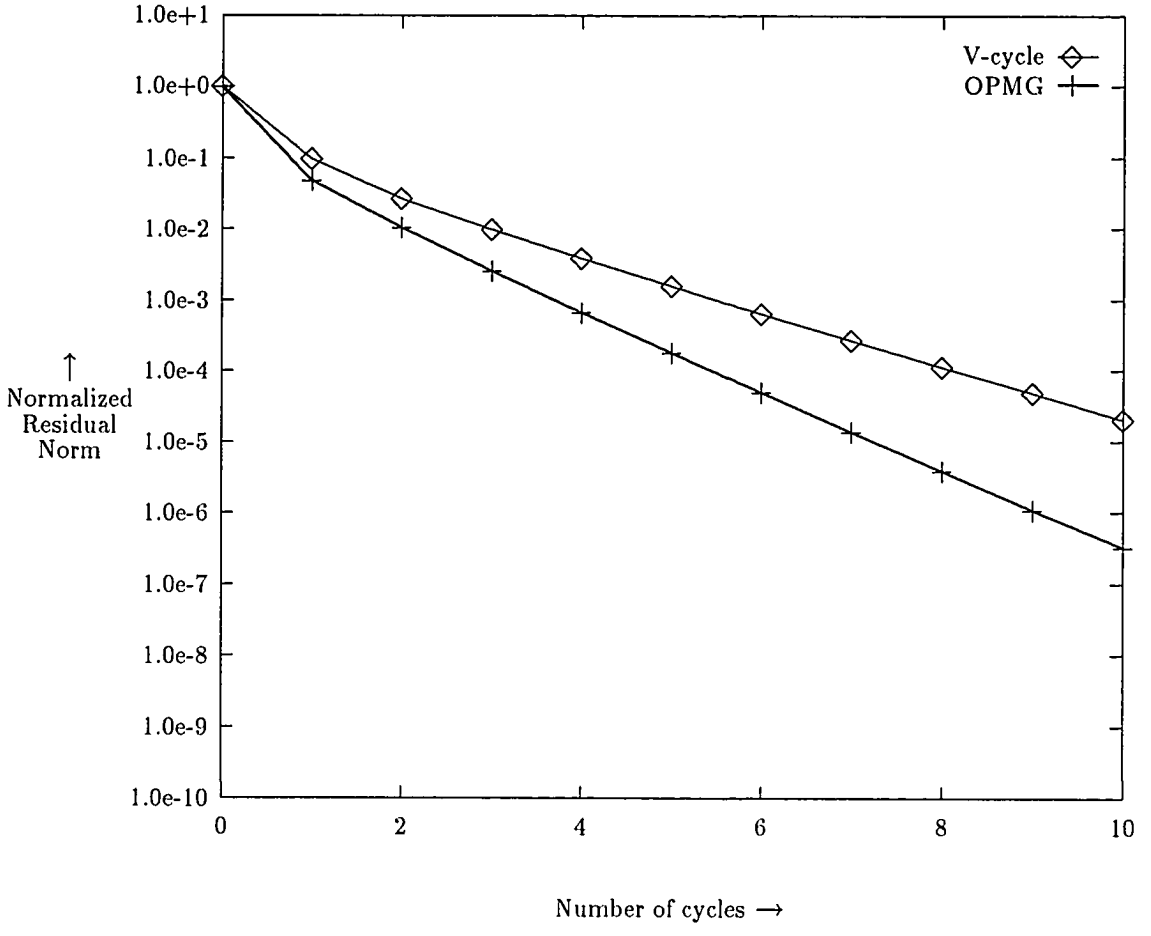


Figure 3.3 : Comparison of normalized residual norm for Problem 1

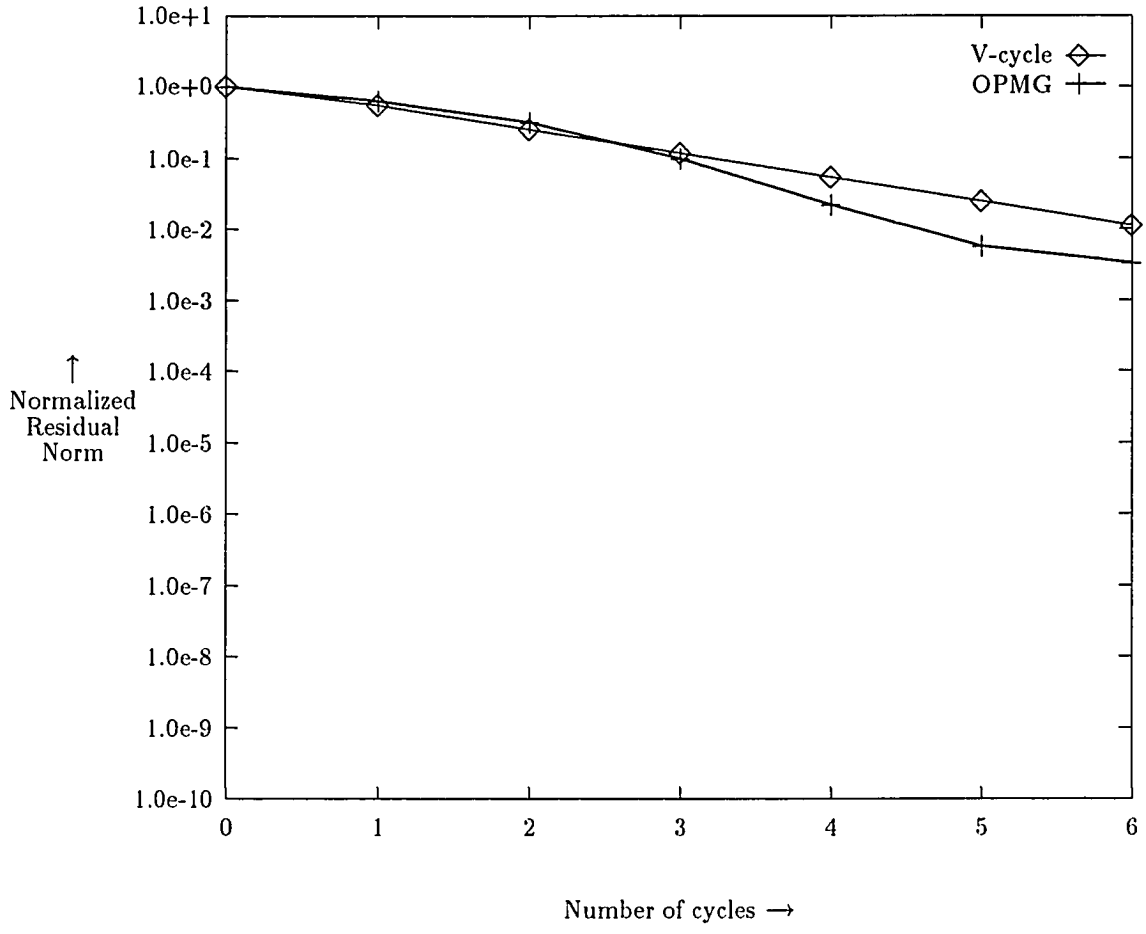


Figure 3.4 : Comparison of normalized residual norm for Problem 2

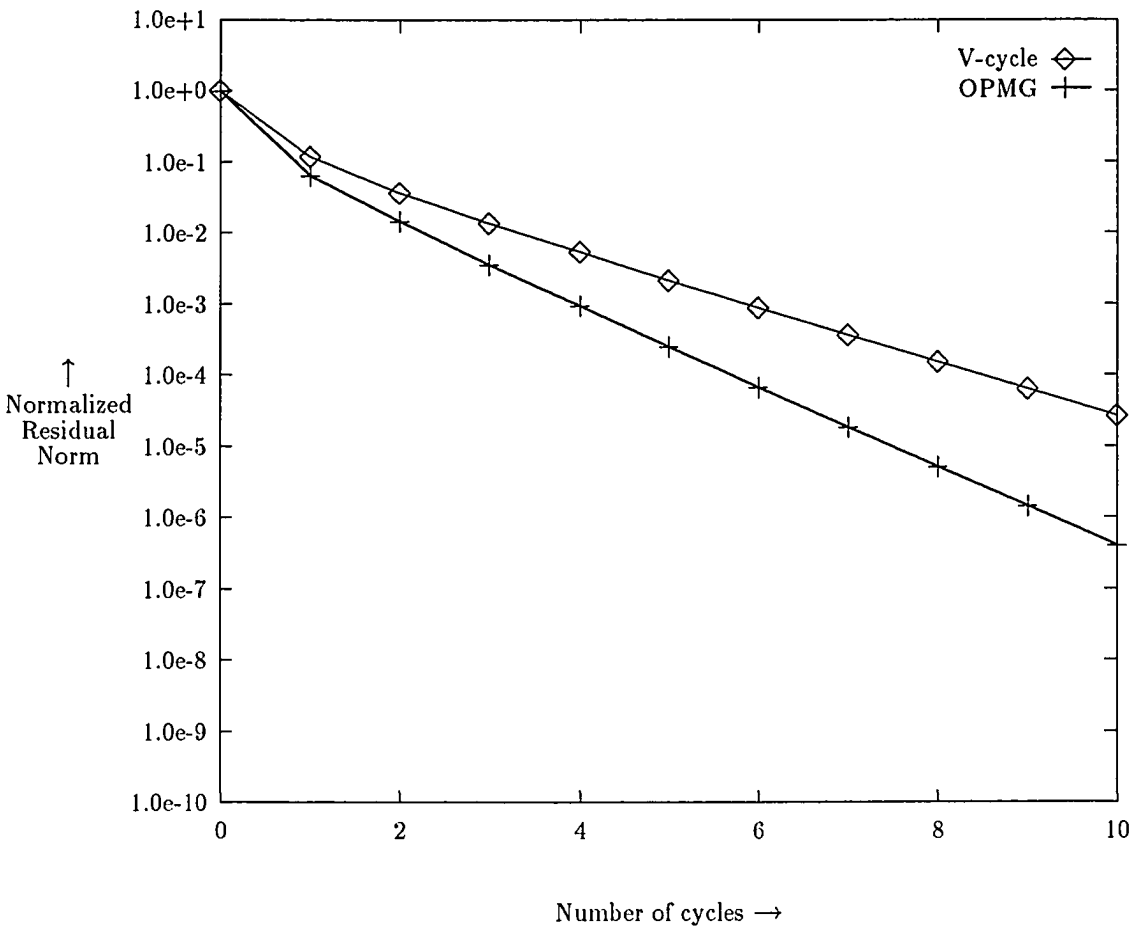


Figure 3.5 : Comparison of normalized norm for Problem 3

three test problems. In this table, we compare the average reduction in the residual norm per cycle for the *OPMG* algorithm and the V-cycle multigrid algorithm. The average reduction in the residual norm per cycle is given by

$$\beta = (\frac{F_{res}}{I_{res}})^{1/n_{cycles}}$$

where

- β is the average reduction in the residual norm per cycle.
- I_{res} is the initial residual norm.
- F_{res} is the final residual norm.
- n_{cycles} is the number of cycles to reduce the residual norm from I_{res} to F_{res}.

Note, that a smaller value of β implies a better convergence rate. Rows 1 and 2 of the table give I_{res} and F_{res} for the V-cycle multigrid algorithm. Row 3 gives the total reduction in the residual norm by taking the ratio of the final residual norm to the initial residual norm. The number of cycles required to reduce the residual norm from I_{res} to F_{res} is given in row 4. In row 5, we give the average reduction in the residual norm per cycle for the V-cycle multigrid algorithm. The next five rows of the table give the same five quantities for the *OPMG* algorithm. Finally, in the last row we give the ratio of the average reduction per cycle in the residual norms for the V-cycle multigrid and the *OPMG* algorithms.

The results of this table show that $\beta(OPMG)$ is smaller than $\beta(V-cycle)$ in all the cases. The ratio of $\beta(V-cycle)$ to $\beta(OPMG)$ indicates that the convergence

Table 3.5 : Summary of simulation results

		Problem 1	Problem 2	Problem 3
V-cycle	I_{res}	1.28	3.76E-5	1.73E-2
	F_{res}	2.60E-5	1.18E-7	4.63E-7
	$\frac{F_{res}}{I_{res}}$	2.03E-5	3.14E-3	2.68E-5
	n_{cycles}	10	10	10
	β	0.339	0.562	0.349
OPMG	I_{res}	1.28	3.76E-5	1.73E-2
	F_{res}	4.22E-7	1.18E-7	7.08E-9
	$\frac{F_{res}}{I_{res}}$	3.30E-7	3.14E-3	4.09E-7
	n_{cycles}	10	7	10
	β	0.225	0.439	0.230
$\frac{\beta(V-cycle)}{\beta(OPMG)}$		1.51	1.28	1.52

rate of the *OPMG* algorithm is better than that of the V-cycle multigrid algorithm by a factor ranging from 1.28 to 1.52 for the three test problems. Thus, we conclude that the *OPMG* algorithm achieves a better rate of convergence than the V-cycle multigrid algorithm because of the extra fine grid computation.

3.4 Implementation of *OPMG* Algorithm on AMT-DAP

In Section 3.1, we briefly described how the *OPMG* algorithm can be implemented on massively parallel SIMD machines without increasing the parallel execution time due to extra computation. Here, we describe the implementation of the *OPMG* algorithm in details for a massively parallel SIMD machine AMT-DAP. The architecture of AMT-DAP is described in Chapter 2.

3.4.1 Mapping of Grids on Processors

The mapping of the grids onto the processors is done in a natural manner for implementation of the *OPMG* algorithm. Assuming that the finest grid has as many grid points as the number of processors, each grid point of this grid is mapped onto one processor. More precisely, a grid point $(i,j), 0 \leq i,j < n$ is mapped onto the processor $p(i,j)$ of the mesh. Here n is the number of grid points in each dimension. Processor assignment to the grid points of the finest grid is done such that the nearest neighbor grid points are assigned to the nearest neighbor processors (Figure 3.6).

The grid points of the next coarse grid are mapped onto alternate processors, both in the X-direction and the Y-direction as shown in Figure 3.6. That is grid point $(i,j), 0 \leq i,j < n/2$ of the next coarse grid is mapped onto processor $p(2*i, 2*j)$ of the mesh. The processors corresponding to nearest neighbor grid points in this case are two distance apart. The other grids in the multigrid hierarchy are mapped in a similar fashion. We have chosen this mapping because it reflects the physical relationship between the grids and it does not incur high overheads during data transfer between two adjacent grids in the hierarchy of grids.

3.4.2 Implementation

The most important aspect of implementing the *OPMG* algorithm on AMT-DAP is overlapping the extra relaxation iteration on a grid with the computation on the next coarse grid. For the sake of clarity, instead of describing the complete implementation of the *OPMG* algorithm, we describe only the overlapping of computation with the help of an example involving the finest grid and the next coarser grid. The overlapping of computation on other grids is done in the same way. First we give the temporal relationship between the operations on the finest grid and the next coarse grid *without* the extra iterations of relaxation. In the following discussion, we refer to the finest grid as fine grid and the next coarse grid as coarse grid.

T_1 : Perform relaxation iteration on fine grid (Pre-relax)

T_2 : Compute residual on fine grid and project them to coarse grid.

T_3 : Perform relaxation iteration on coarse grid (Pre-relax)

T_4 : Compute residual on coarse grid and project them to coarser grid.

....

.... : Operations on more coarse grids

....

T_{m-4} : Apply interpolated corrections to coarse grid.

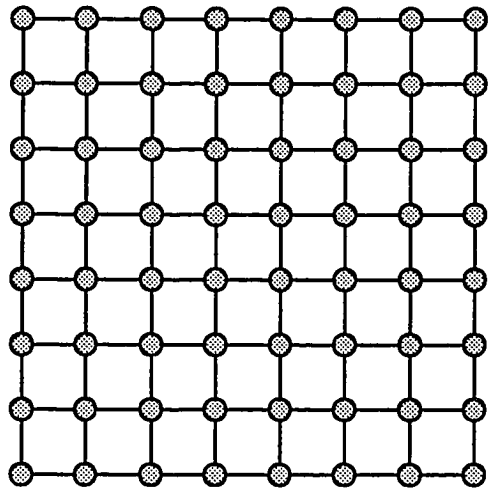
T_{m-3} : Perform relaxation iteration on coarse grid (Post-relax)

T_{m-2} : Interpolate corrections to the fine grid.

T_{m-1} : Apply interpolated corrections to the fine grid.

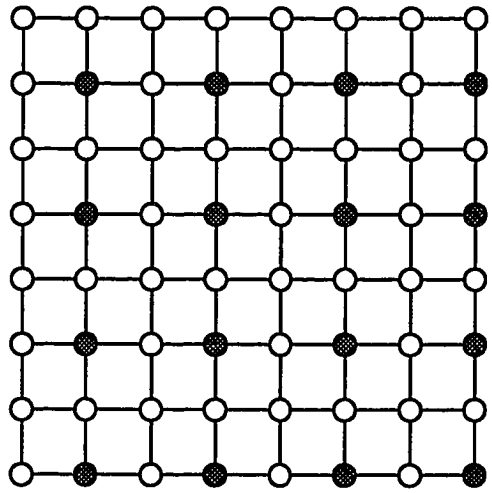
T_m : Perform relaxation iteration on fine grid (Post-relax)

The extra iteration of relaxation on fine grid needs to be done after step T_2 and before step T_{m-1} in the above description. During this period no other operation is being done on the fine grid. Therefore, it is possible to do extra iteration of relaxation on fine grid any time during this period. The SIMD nature of AMT-DAP puts a restriction that the extra iteration of relaxation can be overlapped only with other relaxation steps. In the above description, only steps T_3 and T_{m-3} are relaxation steps. Since the computation on the coarse grid during these steps do not use all the processors, it is possible to do part of the computation associated with the extra iteration of relaxation on the fine grid with the pre-relax step (T_3) and the rest with the post-relax step (T_{m-3}) on the coarse grid. For describing how it is done, we first explain the implementation of a single iteration of relaxation on a grid without overlapping, followed by the implementation with overlapping. We use the weighted Jacobi scheme with weight w for relaxation.



Mapping of the finest grid

● A processor with a finest grid point mapped onto it



Mapping of next coarse grid

● A processor with a next coarse grid point mapped onto it

○ A processor with no grid point mapped onto it

Figue 3.6 : Mapping of two grids onto processors

Relaxation on a single grid without overlapping

- **Get values from neighboring grid points**

  For an iteration of relaxation on a grid, each point of the grid requires values from its four neighboring grid points in East, West, North and South directions. Since on the fine grid these values are in the nearest neighbor processors, one step of communication in each direction is required to obtain these values. For doing the same for the next coarse grid, two steps of communication are required, because the processors corresponding to the nearest neighbor grid points are two distance apart. Assuming that v_{old} is the current value of the approximate solution on the grid points of the fine grid, the values of their four nearest neighbors is obtained by executing the following statements on AMT-DAP.

  $$v_{east} = shwp(v_{old}, 1)$$

  $$v_{west} = shep(v_{old}, 1)$$

  $$v_{north} = shsp(v_{old}, 1)$$

  $$v_{south} = shnp(v_{old}, 1)$$

  Notice that for obtaining the values of the neighboring grid points in the East direction, the values of v_{old} are shifted to the West direction. All other values are obtained similarly.

- **Compute new values**

  Using the old value of the approximate solution (v_{old}), value of the forcing function(f) and the values obtained from the neighboring grid points (v_{east}, v_{west}, v_{north}, v_{south}), the new value of the approximate solution (v_{new}) is computed on each grid point according to the following expression.

$$v_{new} = (1.0 - w) * v_{old} + \frac{w}{4} * (f + v_{east} + v_{west} + v_{north} + v_{south})$$

  There is no communication required in this step because all the values required from other processors have been obtained in the previous step. This operation is exactly the same for both coarse and fine grids except that some of the processors in the coarse grid case do not perform any useful computation (no coarse grid point is mapped onto these processors).

Overlapping of fine grid and coarse grid relaxation

The mapping of the coarse grid onto the processors of the parallel machine indicate that almost $\frac{3}{4}$ of the processors are unutilized during the relaxation computation on the coarse grid. We use these processors to perform the extra iteration of relaxation on the fine grid. Since the relaxation on the fine grid requires all the processors, it is not possible to complete the entire computation of the extra iteration concurrently with one iteration of relaxation on the coarse grid. Therefore, we overlap part of this computation with the pre-relax step and the rest with the post-relax step on the coarse grid. The following is a detailed description of this overlapping.

Pre-relax Step

- **Merge**

  In SIMD machines with no indirect addressing, such as AMT-DAP, it is not possible that some of the processors work on the coarse grid points and the others on the fine grid points. Therefore, we merge the data for the coarse grid and the fine grid to generate a merge grid. In the merge grid, processors used by the coarse grid points retain their data, whereas the unused processors get the fine grid data (Figure 3.7). In the merge grid, points corresponding to the coarse grid have all four of their neighboring grid points, but the same is not true for points corresponding to the fine grid. The status of the fine grid points in the merge grid is shown in Figure 3.8. In this figure, fine grid points marked by a cross are not represented in the merge grid because the corresponding processors are occupied by the coarse grid points. The rest of the fine grid points are represented on the merge grid. Out of these fine grid points, some have all the four neighboring grid points in the merge grid (enclosed by a circle). The others either have only East-West neighbors (enclosed by a horizontal rectangle) or have only North-South neighbors (enclosed by a vertical rectangle).

- **Get values from neighboring grid points**

  The coarse grid points need two steps of communication in each direction to get values of their neighboring grid points whereas the fine grid points need only one step of communication to do the same. Therefore, in the first step

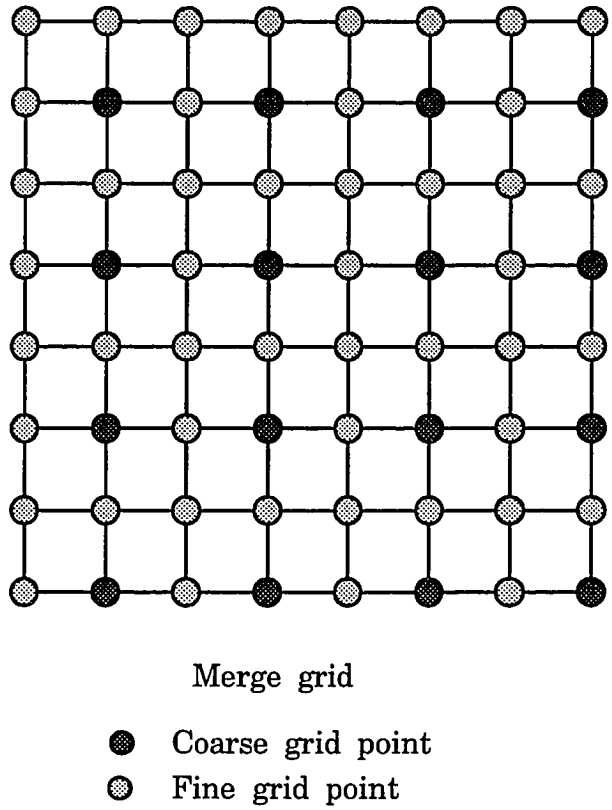


Figure 3.7 : Merge grid showing data from fine and coarse grids

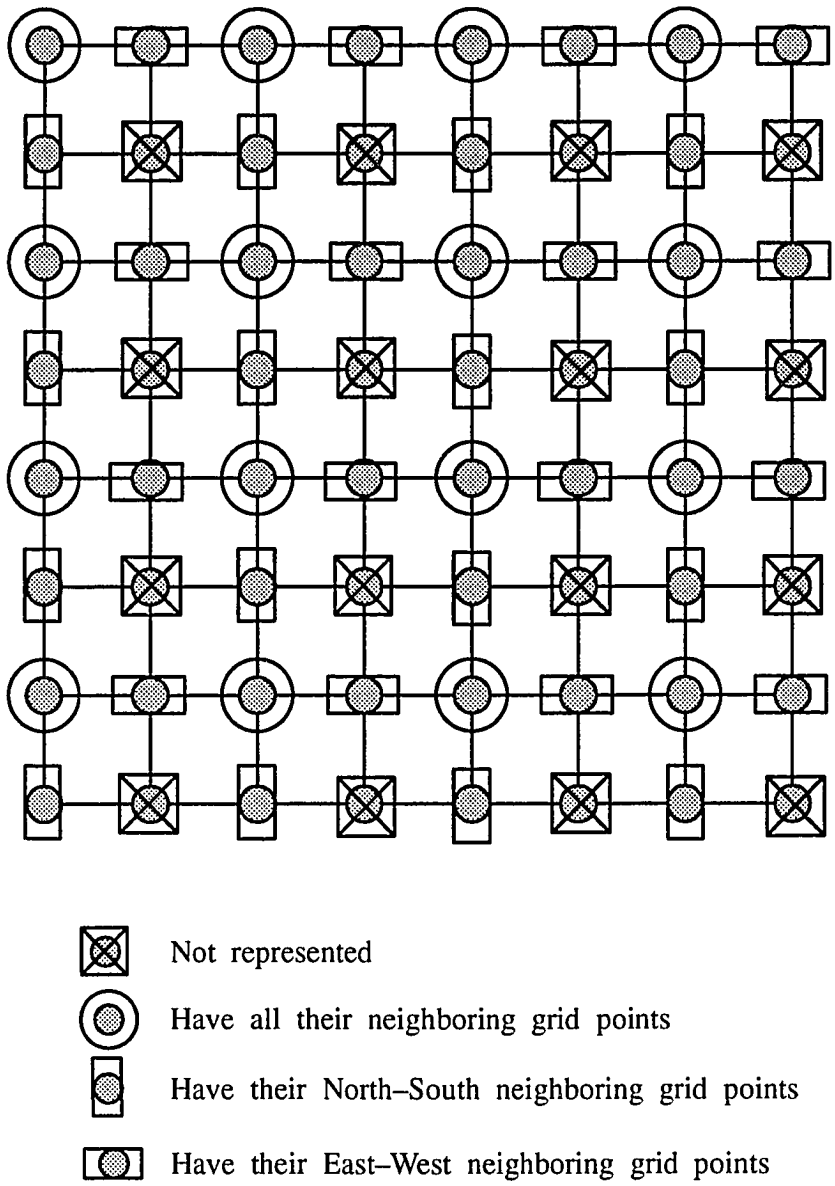
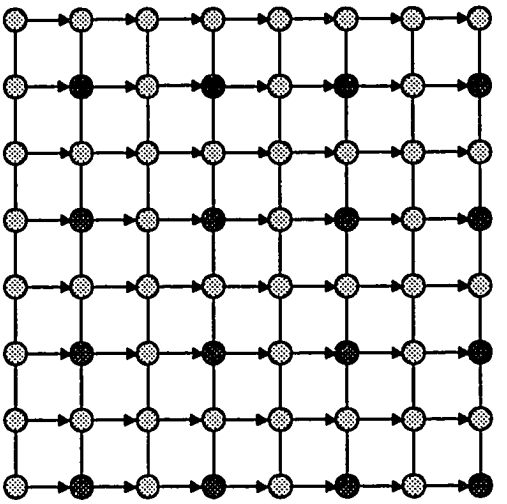


Figure 3.8 : Status of fine grid points in the merge grid

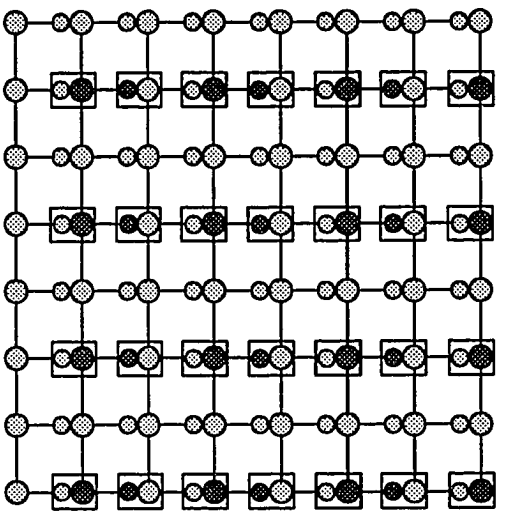
of communication, we shift values on all the processors in the desired direction. This gives the correct values for the fine grid points wherever possible. Figure 3.9 shows the first step for obtaining the values from the neighboring grid points in the West direction. Notice that all the coarse grid points and some of the fine grid points do not have correct values of their neighboring grid points (enclosed by a rectangle). The correct values for the coarse grid points are obtained in step 2, which shifts the values in the desired direction by one. In this step, we mask rows of processors which have the correct values for the fine grid points (Figure 3.10). Even after step 2, some of the fine grid points do not have the correct values of their West neighbors. These are the points for which their West neighbors are not represented on the merge grid (Figure 3.8). The two steps for obtaining the values of the neighboring points in the North direction is shown in Figure 3.11 and Figure 3.12.

- **Compute**

  After the communication step, all the coarse grid points have the values of their four nearest neighbors, but many fine grid points do not have all the four values. Since, relaxation on a grid point requires values of all the four neighboring grid points, we mask the processors corresponding to these fine grid points during relaxation. On these grid points, the values obtained from the two neighboring grid points are saved for later use during the post-relax step.



Step 1 : Shift all values on merge grid in East direction by one.



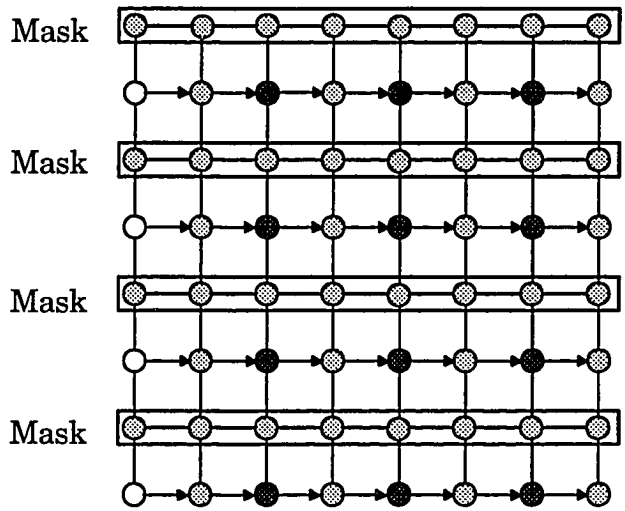
Values on processors after Step 1.

O Original grid point values

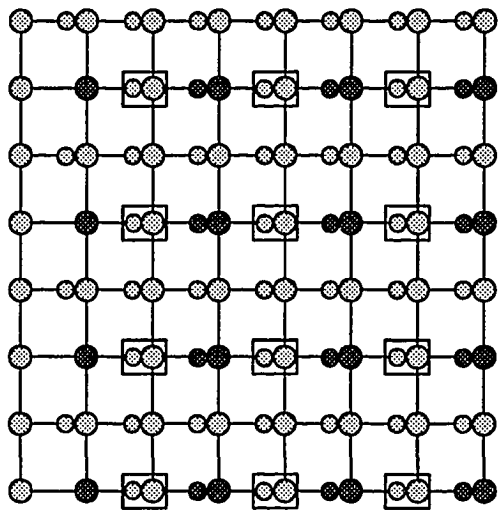
o Values obtained after shifting.

□ Processors which do not have required value of the West neighbor.

Figure 3.9 : Step 1 for obtaining values from the West neighbors



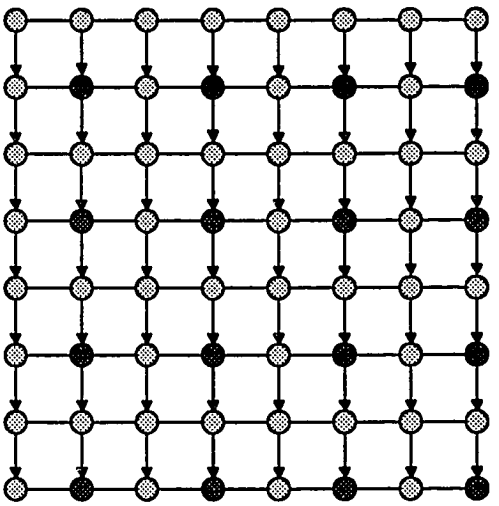
Step 2 : Shift only unmasked rows in East direction by one
(The rows which have obtained required values have been masked.)



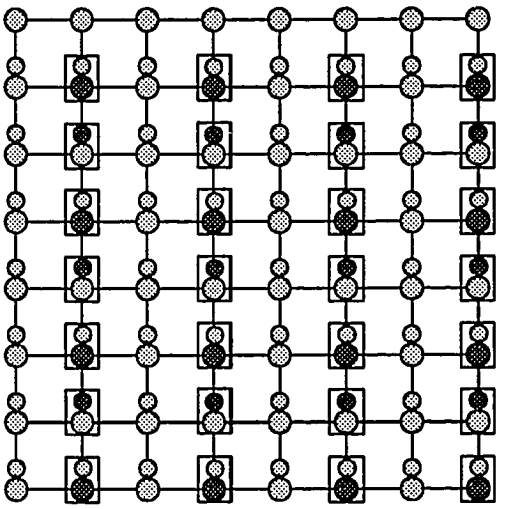
Grid values on processors after step 2.

Processors which do not have
required value of the West neighbor.

Figure 3.10 : Step 2 for obtaining values from the West neighbors



Step 1 : Shift all values on merge grid in the South direction by one.



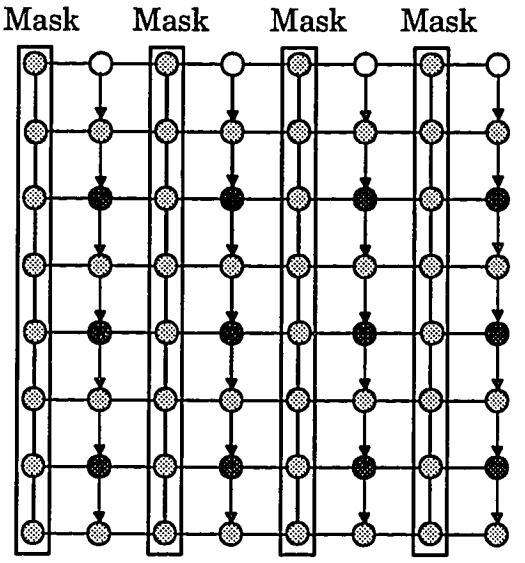
Grid values on processors after Step 1.

O Original grid point values

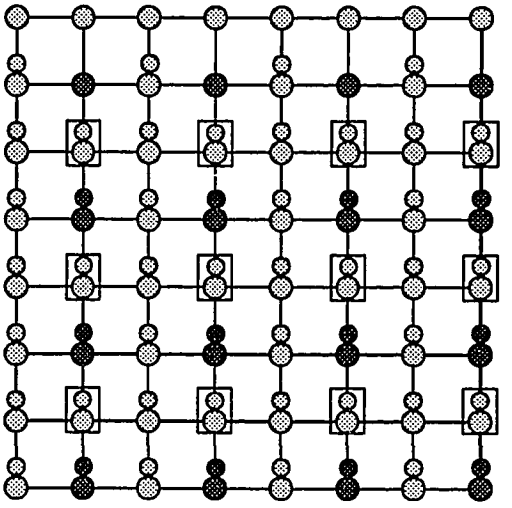
o Values obtained after shifting.

□ Processors which do not have required value of the North neighbor.

Figure 3.11 : Step 1 for obtaining values of the North neighbors



Step 2 : Shift only unmasked rows in the South direction by one
(The rows which have obtained required values have been masked.)



Grid values on processors after Step 2.

Processors which do not have required value of the North neighbor.

Figure 3.12 : Step 2 for obtaining values from the North neighbors

- **Unmerge**

  This step reverses the merging operation by assigning the computed values of the solution back to the coarse and the fine grid. For the fine grid, only those grid points get the values which have completed their computation (enclosed by a circle in Figure 3.8).

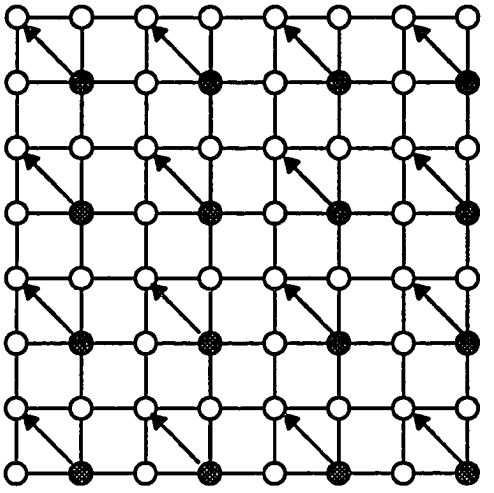
Post-relax Step

- **Merge**

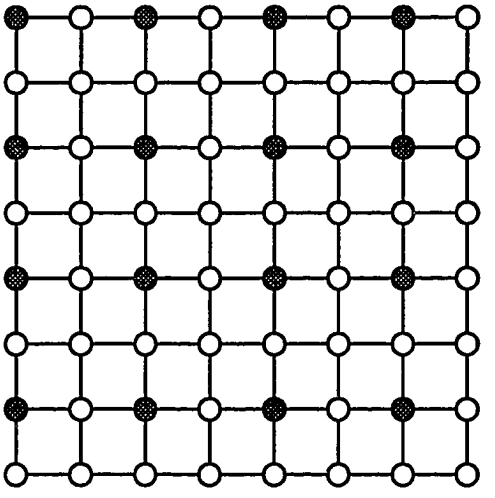
  In the pre-relax step, relaxation iteration could not be completed on all the fine grid points because in the merge grid some of the fine grid points were not present and some others did not have all the values required from neighboring fine grid points. If the merge grid in the post-relax step is produced in the same way, relaxation on these fine grid points will still not be complete. Therefore, we shift the coarse grid diagonally in the North-West direction before merging it with the fine grid (Figure 3.13). The merge grid produced by the shifted coarse grid and the fine grid is shown in Figure 3.14. Notice that this merge grid contains all the fine grid points on which the relaxation could not be completed during the pre-relax step.

- **Get values from neighboring grid points**

  The values of the neighboring grid points are obtained in the same way as in the pre-relax step.



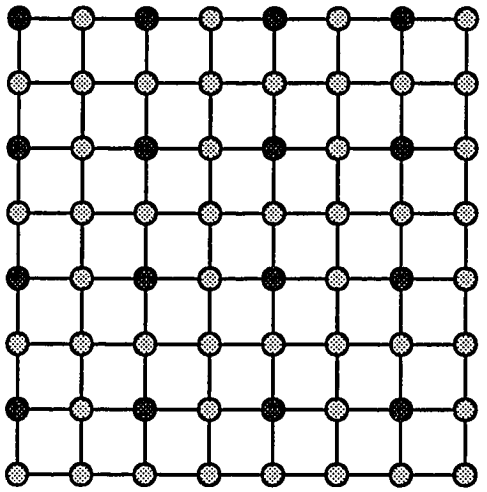
Original mapping of next coarse grid



Mapping of next coarse grid after shifting diagonally

● A processor with a next coarse grid point mapped onto it

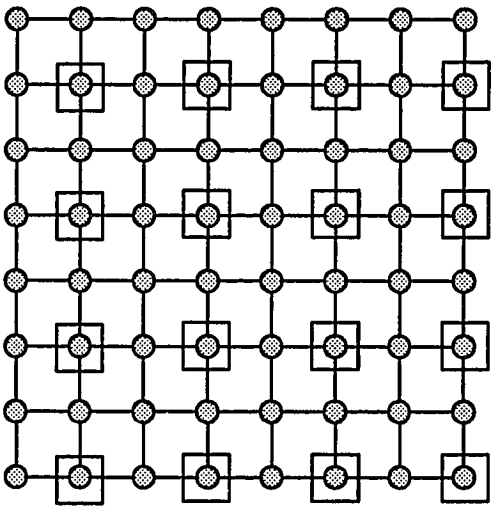
Figure 3.13 : Mapping of coarse grid for post–relax step



Merge grid

● Coarse grid point

◎ Fine grid point



Fine grid

◎ Fine grid points which were not present in the merge grid during pre−relax step (Figure 3.9)

Figure 3.14 : Merge grid for post−relax step

- **Merge the unused values from pre-relax step**

  As in the pre-relax step, some of the fine grid points have values of only two neighboring grid points. To obtain the two remaining values, recall that we saved some values of neighboring grid points in the pre-relax step. By merging these saved values with the currently available neighboring grid points values, all the fine grid points in the merge grid will have values of all the four neighbors. Appropriate processors are masked during this merging so that the useful values are not destroyed.

- **Compute**

  On all the processors, new values are computed after relaxation. Notice that all the fine grid points which could not complete the extra iteration of relaxation during the pre-relax step have completed it now.

- **Unmerge**

  In this step, newly computed values on the merge grid are assigned back to the coarse and fine grid points. The coarse grid values are shifted in the South-East direction to resume their original position.

It is clear from the above description that the coarse grid relaxation and the extra fine grid iterations of relaxation can be overlapped without increasing the parallel execution time by a large amount. The small overheads are caused by the extra communication steps required for shifting the coarse grid diagonally during the post-relax step. The cost of masking some processors during computation and communication is negligible.

3.5 Performance of *OPMG* Algorithm on Parallel Machines

The parallel performance of the *OPMG* algorithm is evaluated by solving the test problems described in Chapter 2 on a 1024 processor AMT-DAP. We compare the results obtained by our parallel implementation of the *OPMG* algorithm with those of the V-cycle multigrid algorithm by using two performance metrics. The first metric compares the average reduction in the residual norm per parallel time unit. The average reduction in the residual norm per cycle is defined in Section 3.3 and a parallel time unit (*ptu*) is the time required for execution of one cycle of the V-cycle multigrid algorithm on AMT-DAP. This performance metric compares the two algorithms by comparing the convergence achieved in the same amount of time. The second performance metric is the parallel execution time required by the two algorithms to reach the same level of convergence.

The results of our comparison using the first metric are shown in Table 3.6. In this table, the first two rows give the average reduction in the residual norm (β(V-cycle)) and the parallel execution time, in seconds, for one cycle of the V-cycle multigrid algorithm. Using these two quantities, we compute the parallel execution time in terms of *ptu*'s and the average reduction in the residual norm per *ptu* represented by γ, as given in rows 3 and 4 respectively. Notice that the parallel execution time for one cycle of the V-cycle multigrid algorithm in terms of *ptu*'s is always one because of the definition of *ptu*. The next four rows of the table give

the same four quantities for the *OPMG* algorithm. Finally, in the last row we give the ratio of $\gamma(V - cycle)$ to $\gamma(OPMG)$, which gives the advantage of the *OPMG* algorithm over the V-cycle multigrid algorithm on massively parallel machines.

The results of comparison using the second metric are shown in Table 3.7. Here, in the first two rows, we give the number of cycles required by the two algorithms to reach the same level of convergence (V_{cycles} and O_{cycles}). In the next two rows, we give the total execution time required by the two algorithms to reach the same level of convergence (V_{total} and O_{total}). These times are computed by multiplying the number of cycles with the execution time for one cycle. The execution time for one cycle of the two algorithms is given in the previous table. Finally, in row 5, we compute the speed-up, which is given by

$$speed - up = \frac{V_{total}}{O_{total}}$$

The results in Table 3.6 and Table 3.7 indicate that the performance of the *OPMG* algorithm on AMT-DAP is better than the performance of the V-cycle multigrid algorithm using both the performance metrics. From Table 3.6 we see the average reduction in the residual norm per *ptu* is approximately 25% to 35% more for the *OPMG* algorithm in comparison to the V-cycle multigrid algorithm. Similarly, the results in Table 3.7 show a speed-up of approximately 1.3. This implies that the *OPMG* algorithm can achieve a solution of the test problems 30% faster than the V-cycle multigrid algorithm. Here, notice that the time required for one cycle of the *OPMG* algorithm is slightly more (10%) than that of the V-cycle multigrid

Table 3.6 : Parallel implementation results for the *OPMG* algorithm (Performance metric 1)

		Problem 1	Problem 2	Problem 3
V-cycle	β	0.342	0.412	0.342
	Execution Time (sec.)	2.21E-2	2.21E-2	2.21E-2
	Execution Time (ptu)	1.0	1.0	1.0
	γ	0.342	0.412	0.342
OPMG	β	0.225	0.304	0.230
	Execution Time (sec.)	2.40E-2	2.40E-2	2.40E-2
	Execution Time (ptu)	1.08	1.08	1.08
	γ	0.251	0.332	0.256
$\frac{\gamma(V-cycle)}{\gamma(OPMG)}$		1.36	1.23	1.34

Table 3.7 : Parallel implementation results for the *OPMG* algorithm (Performance metric 2)

	Problem 1	Problem 2	Problem 3
O_{cycles}	7	7	7
V_{cycles}	10	10	10
O_{total}	1.68E-1	1.68E-1	1.68E-1
V_{total}	2.21E-1	2.21E-1	2.21E-1
$speed - up$	1.31	1.31	1.31

algorithm because of small overheads in overlapping the extra fine grid computation with the coarse grid computation.

3.6 Conclusions

In this chapter, we have presented the *OPMG* algorithm for massively parallel computers. Using analytical and simulation results we have shown that the *OPMG* algorithm achieves a better rate of convergence than the V-cycle multigrid algorithm. The performance of the *OPMG* algorithm on massively parallel machine AMT-DAP shows a speed-up of approximately 30% over the V-cycle multigrid algorithm.

Chapter 4
Chopped Parallel Multigrid Algorithm

4.1 Introduction

In this chapter, we describe the *Chopped Parallel Multigrid* (*CPMG*) algorithm, which is based on the second approach of our classification given in Chapter 1 (Figure 1.4) This approach addresses the problems of low processor utilization and high communication overheads by reducing the amount of work done in each cycle of the multigrid algorithm. For this approach to be effective, the reduction in the work should be done such that the convergence rate of the algorithm is not significantly affected. The *CPMG* algorithm tries to achieve this objective by reducing the amount of work done on the coarse grids. Since the problems of low processor utilization and communication overheads assume greater significance on the coarse grids, reducing the work on these grids relative to the fine grids will help in alleviating these problems.

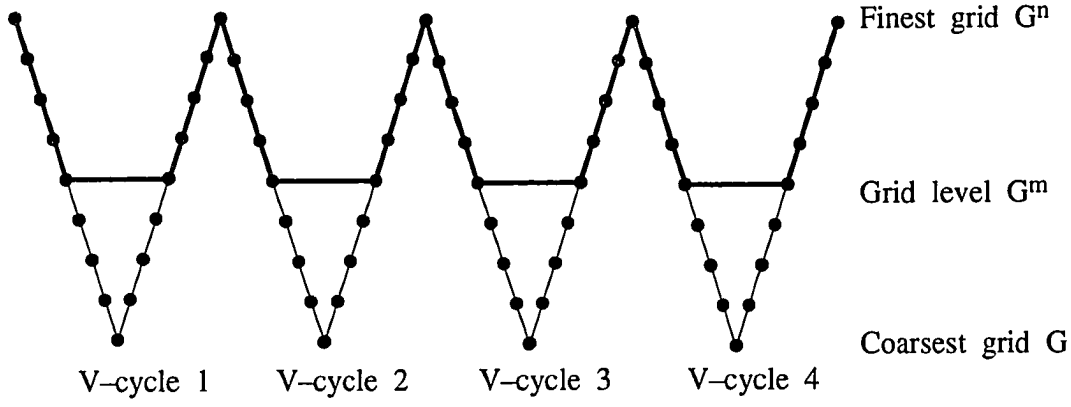


Figure 4.1 : Naive approach of chopping V–cycles
(every V–cycle is chopped)

One naive way to reduce the work on the coarse grids is by *not doing any work* on some of the coarse grids. In other words, during each V-cycle instead of going up to the coarsest grid (G^1), go up to some less coarse grid $G^m, m > 1$(Figure 4.1). We refer to this elimination of some of the coarse grids as the *chopping* of a V-cycle. This naive approach improves the processor utilization and also reduces parallel execution time per cycle but severely degrades the convergence rate in comparison to the V-cycle multigrid algorithm. Similar observation was made by McBryan[24]. The degradation in the convergence rate in this case is because some of the coarse grids on which the very low frequency components of the error were reduced are no longer used. Since relaxation on the rest of the grids cannot remove these error components efficiently, the convergence rate drops significantly. To get around this problem, we refined the naive idea of chopping in the *CPMG* algorithm. The *CPMG* algorithm chops alternate V-cycles as opposed to every V-cycle in the naive approach. That is, the odd numbered cycles use all the grids, whereas the even numbered cycles do not use some of the coarse grids (Figure 4.2). Chopping alternate cycles does not degrade the convergence rate significantly because the very low frequency components of the error which are not removed effectively in a chopped cycle, are removed in the next full cycle. Later in this chapter, we show that the convergence rate of the *CPMG* algorithm is almost the same as that of the V-cycle multigrid algorithm for many cases.

Now, we describe the *CPMG* algorithm more precisely. We use the same notations as those used for describing the V-cycle multigrid algorithm in Chapter 2.

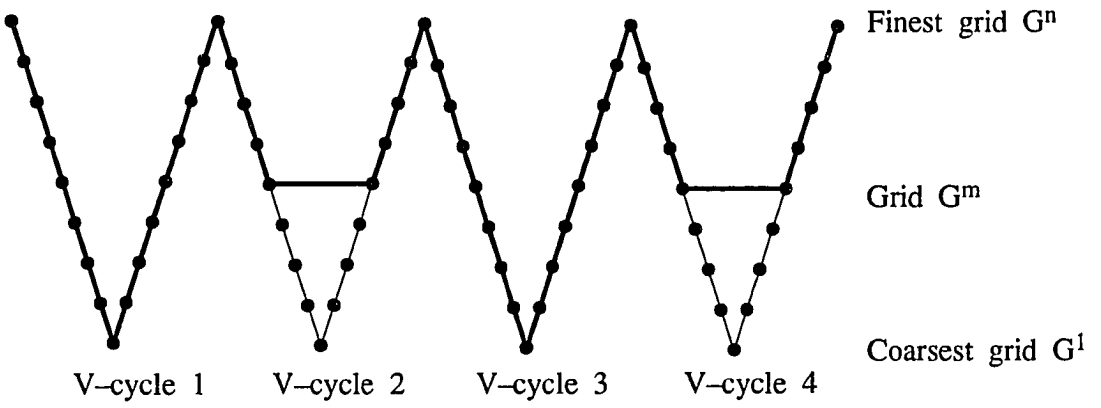


Figure 4.2 : CPMG approach of chopping V–cycles
(alternate V–cycle is chopped)

The parameter m gives the grid level at which alternate V-cycles are chopped.

Algorithm **CPMG** (L^i, f^i, v^i, i, s, m)

begin

$cycle\_no = 1$

while not(converged) **do**

begin

if odd($cycle\_no$) **then**

begin

$coarsest = 1$

Call **MG**$(L^i, f^i, v^i, i, s, coarsest)$

end

else

begin

$coarsest = m$

Call **MG**$(L^i, f^i, v^i, i, s, coarsest)$

end

$cycle\_no = cycle\_no + 1$

end

end

4.2 Convergence Analysis of *CPMG* Algorithm

As mentioned in Chapter 3, the convergence analysis of the multigrid algorithm is usually done with a two grid model, which closely approximates the behavior of the V-cycle multigrid algorithm. However, the analytical comparison of the *CPMG* algorithm and the V-cycle multigrid algorithm using this model will not yield any meaningful results, because in the two grid model a chopped cycle is identical to a normal cycle. Therefore, we must use an all grid model for our analysis. However, this creates a problem of expressing the error in a closed form expression. We overcome this by simulating the all grid model.

The two algorithms are compared by computing the reduction in the magnitude of different frequency components of the error by each algorithm. This is done by deriving expressions for error reduction due to the four basic operations (relaxation, residual computation, projection and interpolation) on each grid. These expressions are then used in a computer program to compute the net reduction in error by all the grids in the multigrid hierarchy. Our analysis follows the approach described in [2].

We use the following one-dimensional model problem for analysis.

$$u''(x) = f, \qquad 0 < x < 1$$

$$u(0) = u(1) = 0$$

The finite difference discretization of this problem using a one dimensional grid of

$(N+1)$ points results in a system of linear equations.

$$A\mathbf{u} = \mathbf{f}$$

For simplicity, we assume $N = 2^n$, where n is an integer. The matrix A has a tridiagonal structure with all of the diagonal elements equal to 2 and all the super-diagonal and sub-diagonal elements equal to -1. The eigenvalues λ_k and eigenvectors w_k of the matrix A are,

$$\begin{aligned} \lambda_k(A) &= 4sin^2\left(\frac{k\pi}{2N}\right), \quad 1 \leq k \leq N-1 \\ w_{k,j} &= sin\left(\frac{jk\pi}{N}\right), \quad 1 \leq k \leq N-1, \quad 0 \leq j \leq N \end{aligned}$$

Where, $w_{k,j}$ is the jth component of kth eigenvector w_k. Since the eigenvectors of the matrix A are the same as the Fourier modes of the error [2], we use w_k to represent the kth Fourier mode of the error also. Note, that previously we have been referring to Fourier modes of the error as frequency components of the error.

Now we compute expressions for error reduction due to the four basic operations. In the following we use super-scripts to refer to the grid level unless specified otherwise.

1. Relaxation : For relaxation on a grid level i we use the weighted Jacobi method with weight equal to $1/2$. If the initial error is w_k^i, then the error after one iteration of relaxation (e_k^i) is

$$e_k^i = S^i w_k^i,$$

where S^i is the iteration matrix for the weighted Jacobi relaxation scheme and is given by the following equation for weight equal to 1/2.

$$S^i = I - \frac{1}{4}A^i$$

The eigenvectors of S^i are the same as the eigenvectors of the matrix A^i and the eigenvalues are

$$\lambda_k(S^i) = 1 - \frac{1}{4}\lambda_k(A^i)$$

Using the fact that w_k^i is the eigenvector of S^i also, we can simplify the expression for the new error (e_k^i).

$$\begin{aligned} e_k^i &= S^i w_k^i \\ &= \lambda_k(S^i).w_k^i \\ &= (1 - \frac{1}{4}\lambda_k(A^i))w_k^i \\ &= (1 - sin^2\left(\frac{k\pi}{2N^i}\right))w_k^i \\ &= cos^2\left(\frac{k\pi}{2N^i}\right)w_k^i \end{aligned}$$

2. Residual Computation : The residual r_k^i corresponding to a error w_k^i is given by

$$\begin{aligned} r_k^i &= A^i w_k^i \\ &= \lambda_k^i w_k^i \\ &= 4sin^2\left(\frac{k\pi}{2N^i}\right)w_k^i \end{aligned}$$

3. Projection (P^{i-1}) : The projection operator used is a full weighting operator [2]. It projects the kth Fourier mode on grid i to the kth Fourier mode on grid $(i-1)$. It also projects the $(N^i - k)th$ Fourier mode on grid i to the kth Fourier mode on grid $(i-1)$.

$$\begin{aligned} P^{i-1}w_k^i &= cos^2\left(\frac{k\pi}{2N^i}\right)w_k^{i-1}, \quad 1 \le k \le \frac{N^i}{2} \\ P^{i-1}w_{k'}^i &= -sin^2\left(\frac{k\pi}{2N^i}\right)w_k^{i-1}, \quad 1 \le k < \frac{N^i}{2}, \quad k' = N^i - k \end{aligned}$$

4. Interpolation (I^i): The operator used for interpolation from a coarse grid to a fine grid is a bilinear interpolation operator [2]. Interpolation of the kth mode from grid $(i-1)$ results in two modes on grid i. These are given by

$$\begin{aligned} I^i w_k^{i-1} &= cos^2\left(\frac{k\pi}{2N^i}\right)w_k^i - sin^2\left(\frac{k\pi}{2N^i}\right)w_{k'}^i, \\ & 1 \le k \le \frac{N}{2}, \quad k' = N^i - k \end{aligned}$$

We use the above four operators to compute the effect of the V-cycle multigrid algorithm and the *CPMG* algorithm on different Fourier modes of the error. Even though we start with a single error mode, both algorithms give rise to several other modes due to the projection and the interpolation operations. In order to compute the net reduction in the error, we take the $\|.\|_2$ norm of the magnitude of all the resulting Fourier modes of error. For describing the results of our analysis, we represent the standard V-cycle multigrid algorithm as $MG(n,s)$ and the *CPMG* algorithm as $CPMG(n,m,s)$, where n is the total number of grids (total number of grid points in the finest grid are 2^n), m is the coarsest grid in the chopped cycle and s is the number of relaxation iterations on each grid.

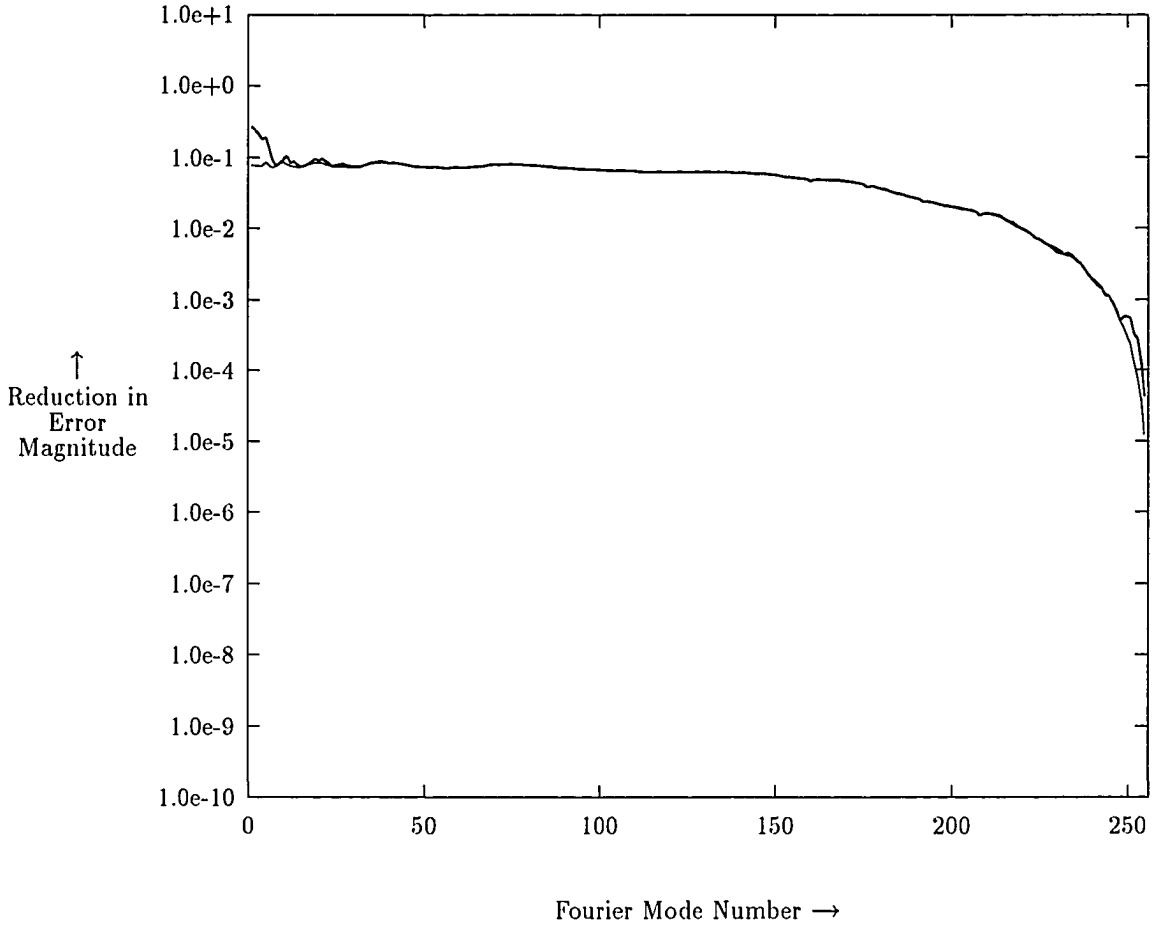


Figure 4.3 : Analytical comparison of $CPMG(8,5,1)$ and $MG(8,1)$

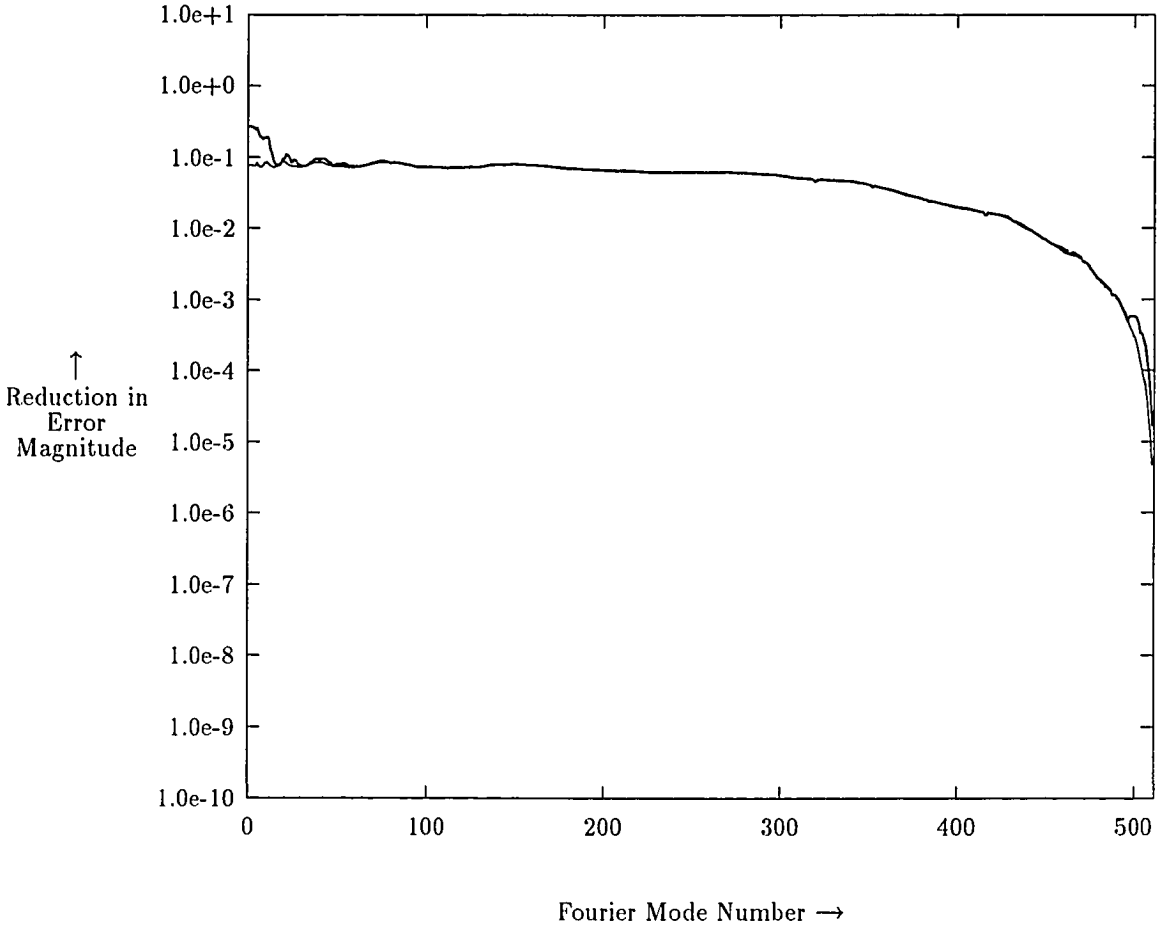


Figure 4.4 : Analytical comparison of $CPMG(9,5,1)$ and $MG(9,1)$

The results from analytical comparison of the two algorithms are shown in Figure 4.3 and Figure 4.4. The curves of these figures show the reduction in the error for each Fourier mode. The *log* of the reduction in the error after two cycles of standard V-cycle algorithm and two cycles of the *CPMG* algorithm (constituting one normal cycle and one chopped cycle) are plotted versus the mode numbers. The graph of Figure 4.3 compares $MG(8,1)$ with $CPMG(8,5,1)$ (only 5 out of the 8 total grids are used in the chopped cycle). It is clear from this graph that the reduction in the error for all the frequency modes is almost the same for the two algorithms except for some of the very low frequency modes. A similar behavior is evident from the graph of Figure 4.4, which compares $MG(9,1)$ and $CPMG(9,5,1)$ (only 5 out of the 9 total grids are used in the chopped cycle). Since the reduction in the individual frequency modes of error is directly related to the convergence rate of the algorithm, we conclude that the convergence rates of the two algorithms are almost identical for many cases. This will be demonstrated in the next section by comparing the convergence rates for a number of problems solved using these two algorithms.

4.3 Simulation Results for *CPMG* Algorithm's Convergence

We simulate the test problems described in Chapter 2, in order to compare the convergence rates of the V-cycle multigrid and the *CPMG* algorithms. Each of

these problems requires solution of the Poisson equation in two dimensions with zero boundary condition and a different forcing function. The finest grid used for discretization in all the cases has 511 points in each dimension, excluding boundary points. The chopped cycle does not use four out of nine grids (*CPMG(9,5,1)*). We use the weighted Jacobi relaxation scheme with weight $\frac{2}{3}$ for all the simulations. Since the purpose of these simulations is to compare the convergence rate, we perform them on a sequential machine. The methodology for our simulations is described in Chapter 2.

The results of our simulations are tabulated in Table 4.1 to Table 4.3. These tables compare the convergence behaviors of the two algorithms in terms of the $\|.\|_2$ norm of residual and also the maximum absolute residual after each cycle. For all the problems convergence of $CPMG(9,5,1)$ is compared against $MG(9,1)$.

From the results shown in these tables it is clear that the convergence rate of $MG(9,1)$ and $CPMG(9,5,1)$ is almost identical for problems 1 and 3. For problem 2 the performance of $MG(9,1)$ is better than that of $CPMG(9,5,1)$. The difference in the convergence rate for problem 2 is due to the fact that the initial error in this case contains only the lowest frequency mode and from the analysis done in Section 4.2, we know that the convergence rates of the two algorithms differ for very low frequency modes.

We have also depicted the results of our simulations in graphs of Figure 4.5 to Figure 4.7. In all these graphs, the number of cycles is plotted on the X-axis and the normalized reduction in the $\|.\|_2$ norm of residual is plotted on the Y-axis. The

Table 4.1 : Convergence history comparison for Problem 1

No. of cycles	V-cycle residual norm	*CPMG* residual norm	V-cycle max residual	*CPMG* max residual
0	1.28E+0	1.28E+0	3.76E+0	3.76E+0
1	1.22E-1	1.22E-1	5.91E-1	5.91E-1
2	3.40E-2	3.40E-2	2.22E-1	2.22E-1
3	1.25E-2	1.25E-2	8.92E-2	8.92E-2
4	4.90E-3	4.90E-3	3.69E-2	3.69E-2
5	1.98E-3	1.98E-3	1.53E-2	1.53E-2
6	8.16E-4	8.16E-4	6.48E-3	6.48E-3
7	3.41E-4	3.41E-4	2.78E-3	2.78E-3
8	1.44E-4	1.44E-4	1.20E-3	1.20E-3
9	6.10E-5	6.10E-5	5.16E-4	5.17E-4
10	2.60E-5	2.61E-5	2.23E-4	2.24E-4

Table 4.2 : Convergence history comparison for Problem 2

No. of cycles	V-cycle residual norm	*CPMG* residual norm	V-cycle max residual	*CPMG* max residual
0	3.76E-5	3.76E-5	7.53E-5	7.53E-5
1	2.01E-5	2.01E-5	3.45E-4	3.45E-4
2	8.46E-6	1.43E-5	5.45E-5	8.10E-5
3	3.67E-6	7.14E-6	2.99E-5	4.76E-5
4	1.57E-6	5.30E-6	9.36E-6	2.49E-5
5	6.74E-7	2.67E-6	4.65E-6	1.76E-5
6	3.06E-7	1.96E-6	2.15E-6	8.88E-6
7	1.67E-7	1.00E-6	1.19E-6	6.26E-6
8	1.28E-7	7.32E-7	9.54E-7	3.46E-6
9	1.19E-7	3.89E-7	8.34E-7	2.15E-6
10	1.18E-7	2.93E-7	7.75E-7	1.43E-6

Table 4.3 : Convergence history comparison for Problem 3

No. of cycles	V-cycle residual norm	*CPMG* residual norm	V-cycle max residual	*CPMG* max residual
0	1.73E-2	1.73E-2	6.46E+0	6.46E+0
1	2.02E-3	2.02E-3	7.04E-1	7.04E-1
2	6.27E-4	6.27E-4	1.79E-1	1.79E-1
3	2.32E-4	2.32E-4	5.58E-2	5.58E-2
4	9.10E-5	9.10E-5	1.87E-2	1.87E-2
5	3.66E-5	3.66E-5	6.52E-3	6.52E-3
6	1.50E-5	1.50E-5	2.34E-3	2.34E-3
7	6.21E-6	6.21E-6	8.61E-4	8.61E-4
8	2.60E-6	2.60E-6	3.37E-4	3.37E-4
9	1.09E-6	1.09E-6	1.39E-4	1.39E-4
10	4.64E-7	4.64E-7	5.86E-5	5.85E-5

normalized reduction in the residual norm is computed by taking the ratio of the current residual norm to the initial residual norm.

Finally, we summarize the results of our simulations in Table 4.4 for all the three test problems. In this table, we compare the average reduction in the residual norm per cycle for the *CPMG* algorithm and the V-cycle multigrid algorithm. Rows 1 and 2 of Table 4.4 give the initial residual norm and the final residual norm (I_{res} and F_{res}) for the V-cycle multigrid algorithm. Row 3 gives the total reduction in the residual norm. Number of cycles required by the V-cycle multigrid algorithm to reduce the residual norm from I_{res} to F_{res} is given in row 4. In row 5, we give the average reduction in the residual norm per cycle (β) for the V-cycle multigrid algorithm. Rows 6 to 10 of the table give the same five quantities for the *CPMG* algorithm. Finally, in row 11 we give the ratio of the average reduction per cycle in the residual norms of the V-cycle multigrid and the *CPMG* algorithms. These results show that the reduction in the residual norm per cycle of the two algorithms is the same for problems 1 and 3. For problem 2 the V-cycle multigrid algorithm performs better in terms of the average reduction in the residual norm by 9.0%.

From the results shown in the above tables and graphs, we conclude that the convergence rates of $MG(n,s)$ and $CPMG(n,\lceil\frac{n}{2}\rceil,s)$ (half the grids are not used in the chopped cycle) are almost identical for many cases. For the cases where the initial error is dominated by very low frequency components, the convergence of the V-cycle multigrid algorithm is slightly better than that of the *CPMG* algorithm.

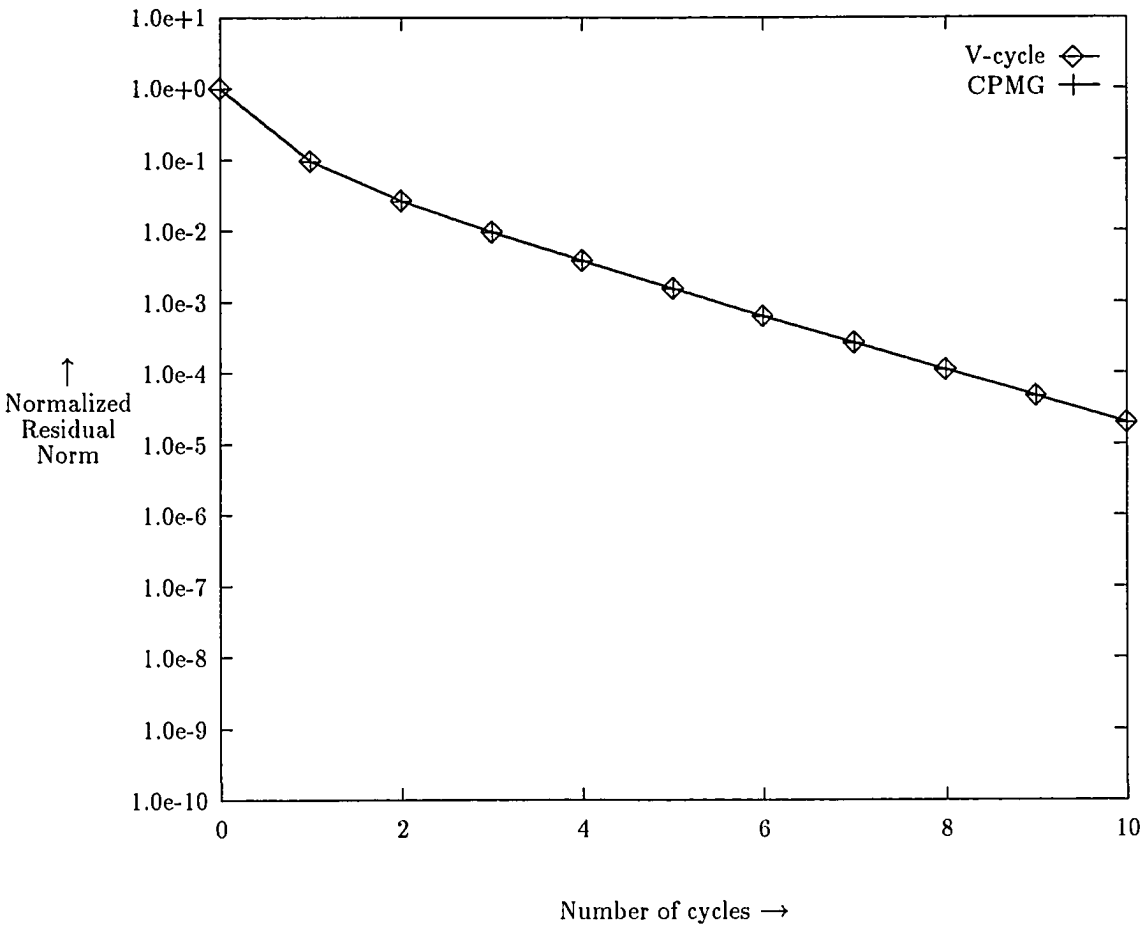


Figure 4.5 : Comparison of normalized residual norm for Problem 1

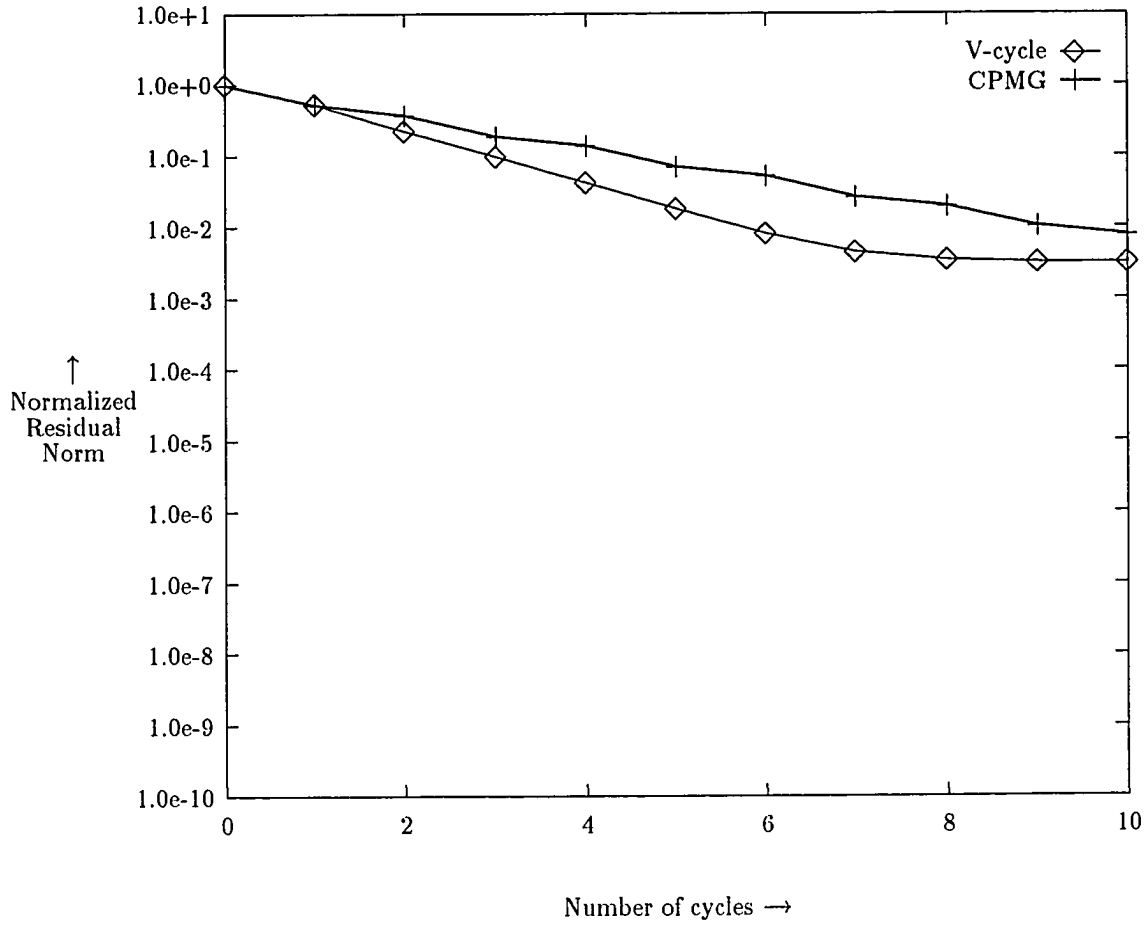


Figure 4.6 : Comparison of normalized residual norm for Problem 2

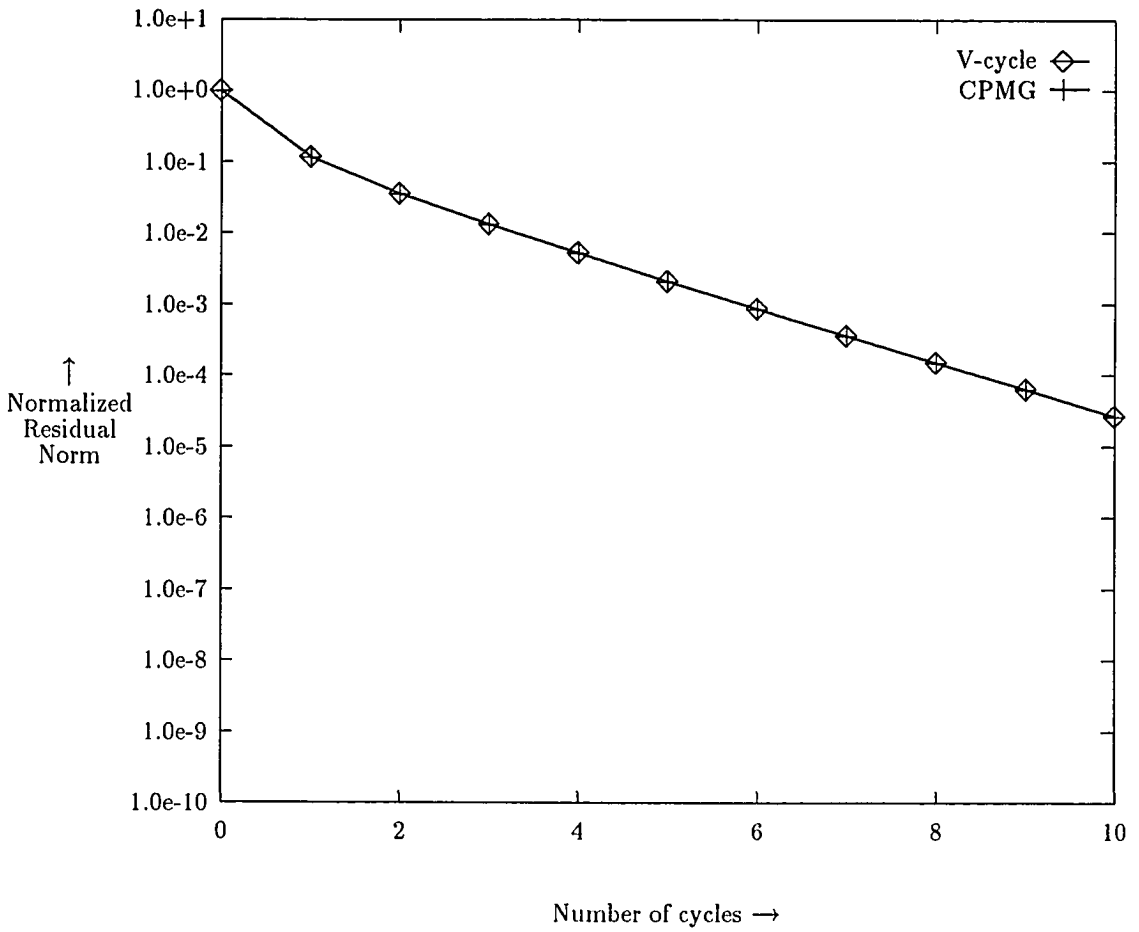


Figure 4.7 : Comparison of normalized residual norm for Problem 3

Table 4.4 : Summary of simulation results

		Problem 1	Problem 2	Problem 3
V-cycle	I_{res}	1.28	3.76E-5	1.73E-2
	F_{res}	2.60E-5	1.18E-7	4.63E-7
	$\frac{F_{res}}{I_{res}}$	2.03E-5	3.14E-3	2.68E-5
	n_{cycles}	10	10	10
	β	0.339	0.562	0.349
CPMG	I_{res}	1.28	3.76E-5	1.73E-2
	F_{res}	2.61E-5	2.93E-7	4.64E-7
	$\frac{F_{res}}{I_{res}}$	2.04E-5	7.79E-3	2.68E-5
	n_{cycles}	10	10	10
	β	0.339	0.615	0.349
$\frac{\beta(V-cycle)}{\beta(CPMG)}$		1.00	0.91	1.00

4.4 Implementation of *CPMG* Algorithm on AMT-DAP

Implementation of the *CPMG* algorithm is similar to the *OPMG* algorithm, details of which are given in Chapter 3. However, in the implementation of the *CPMG* algorithm, the overlapping of grids is not required. The other difference is that some of the grids are not used in alternate cycles of the *CPMG* algorithm.

4.5 Performance of *CPMG* Algorithm on Parallel Machines

The parallel performance of the *CPMG* algorithm is evaluated by solving the test problems described in Chapter 2 on a 1024 processor AMT-DAP. Since the 1024 processors AMT-DAP are arranged in a 32×32 mesh, the finest grid used for these problems has 32 grid points in each dimension. We compare the results obtained by our parallel implementation of the *CPMG* algorithm with those of the V-cycle multigrid algorithm by using the same two performance metrics, used in Chapter 3, for the *OPMG* algorithm. The first metric compares the average reduction in residual norm per parallel time unit (*ptu*) and the second compares the parallel execution time required by the two algorithms to reach the same level of convergence.

The results of our comparison using the first metric are shown in Table 4.5. In this table, we first give the average reduction in the residual norm per cycle

(β(V-cycle)) and the parallel execution time in seconds for one cycle of the V-cycle multigrid algorithm. In the next two rows, we give the parallel execution time in terms of *ptu*'s and the average reduction in the residual norm per *ptu* represented by γ, for the V-cycle multigrid algorithm. The next four rows of the table give the same four quantities for the *CPMG* algorithm. Finally, in the last row, we give the ratio of $\gamma(V-cycle)$ to $\gamma(CPMG)$, which gives the advantage of the *CPMG* algorithm over the V-cycle multigrid algorithm on massively parallel machines. These results show that the *CPMG* algorithm converges approximately 40% faster than the V-cycle multigrid algorithm for problems 1 and 3. For problem 2, the V-cycle multigrid algorithm converges 12% faster.

The results of comparison using the second metric are shown in Table 4.6. Here, in the first two rows we give the number of cycles required by the two algorithms to reach the same level of convergence (C_{cycles} and V_{cycles}). In the next two rows, we give the total execution time required by the two algorithms to reach the same level of convergence (C_{total} and V_{total}). Finally, in row 5 we give the speed-up, as defined in Chapter 3. The results in this table show that the *CPMG* algorithm has an advantage of approximately 30% over the V-cycle multigrid algorithm, in terms of the parallel execution time, for problems 1 and 3. For problem 2, the *CPMG* algorithm's performance is slightly poorer than the V-cycle multigrid algorithm because of its slower convergence rate.

Table 4.5 : Parallel implementation results for the *CPMG* algorithm (Performance metric 1)

		Problem 1	Problem 2	Problem 3
V-cycle	β	0.342	0.412	0.342
	Execution Time (sec.)	2.21E-2	2.21E-2	2.21E-2
	Execution Time (ptu)	1.0	1.0	1.0
	γ	0.342	0.412	0.342
CPMG	β	0.342	0.554	0.342
	Execution Time (sec.)	1.70E-2	1.70E-2	1.70E-2
	Execution Time (ptu)	0.77	0.77	0.77
	γ	0.246	0.464	0.247
$\frac{\gamma(V-cycle)}{\gamma(CPMG)}$		1.39	0.88	1.38

Table 4.6 : Parallel implementation results for the *CPMG* algorithm (Performance metric 2)

	Problem 1	Problem 2	Problem 3
C_{cycles}	10	10	10
V_{cycles}	10	7	10
C_{total}	1.70E-1	1.70E-1	1.70E-1
V_{total}	2.21E-1	1.58E-1	2.21E-1
$speed - up$	1.30	0.92	1.30

4.6 Conclusions

In this chapter, we have described a new parallel multigrid algorithm, the *CPMG* algorithm. We have shown that the *CPMG* algorithm has nearly the same convergence rate as the V-cycle multigrid algorithm for those problems in which the initial error is distributed over all the frequency modes or is concentrated in the high frequency modes. If nearly all the error is concentrated in very low frequency modes, then the *CPMG* algorithm converges slower than the standard V-cycle multigrid algorithm.

We have also shown that for those problems, which do not have all the error concentrated in very low frequency modes, the *CPMG* algorithm shows a significant speed-up over the V-cycle multigrid algorithm. The results of the parallel implementation of the two algorithms on AMT-DAP show a speed-up of 30% for problems 1 and 3 using the *CPMG* algorithm. For problem 2, which is the worst case, the *CPMG* algorithm is 8% slower than the V-cycle multigrid algorithm.

Chapter 5
Hybrid Multigrid Algorithm

5.1 Introduction

The two algorithms presented in the previous chapters, the *OPMG* algorithm and the *CPMG* algorithm use complementary approaches for improving the performance of the multigrid algorithm on parallel machines. The *OPMG* algorithm achieves a better convergence rate than the V-cycle multigrid algorithm while keeping the parallel execution time per cycle the same. This is achieved by doing extra computation on unutilized processors of the parallel machine. In contrast, the *CPMG* algorithm reduces the average parallel execution time per cycle while keeping the convergence rate the same. This is done by reducing the computational work on the coarse grids in comparison to the fine grids. The complementary natures of these approaches suggest that a hybrid algorithm will be more effective in solving the problems associated with the parallel implementation of the multigrid algorithm than either the

OPMG algorithm or the *CPMG* algorithm alone.

In the hybrid algorithm, alternate cycles are chopped similar to the *CPMG* algorithm and extra iterations of relaxation are done similar to the *OPMG* algorithm. The odd numbered cycles of the resulting algorithm are the same as the *OPMG* algorithm. The even numbered cycles also perform the same computation as the *OPMG* algorithm but are chopped as in the *CPMG* algorithm. The combination of the two algorithms in the hybrid algorithm gives a two fold advantage. Firstly, it obtains a better rate of convergence because of the extra computation on the unutilized processors. Secondly, it takes less parallel execution time per cycle because of the chopped cycles. Both these factors reduce the total parallel execution time required to solve a problem.

Now we give a precise description of the hybrid algorithm. We use the same notations as used in the previous chapters to describe the *OPMG* and the *CPMG* algorithms.

Algorithm **HYBRID** $(L^i, f^i, v^i, i, s1, s2, m)$

begin

$cycle\_no = 1$

while not(converged) **do**

begin

if odd($cycle\_no$) **then**

begin

$coarsest = 1$

Call **OPMG**$(L^i, f^i, v^i, i, s1, s2, coarsest)$

end

else

begin

$coarsest = m$

Call **OPMG**$(L^i, f^i, v^i, i, s1, s2, coarsest)$

end

$cycle\_no = cycle\_no + 1$

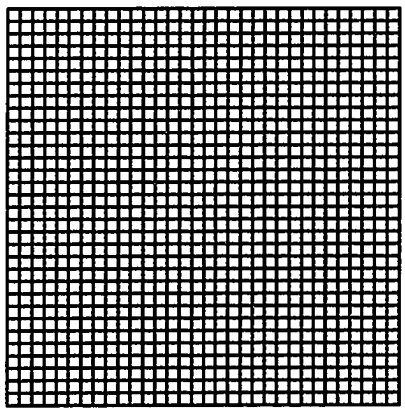
end

end

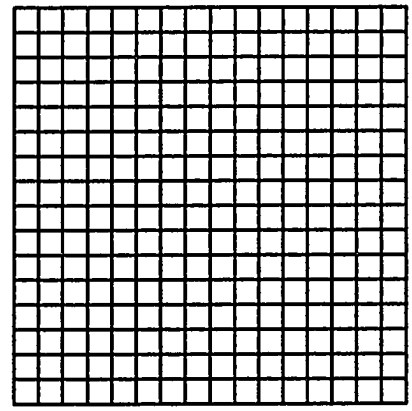
The sequence of operations performed by the hybrid algorithm, for a hierarchy of five grids shown in Figure 5.1, is given in Figure 5.2. In this case, the chopped cycle of the hybrid algorithm uses only three out of the five grids.

5.2 Convergence of Hybrid Algorithm

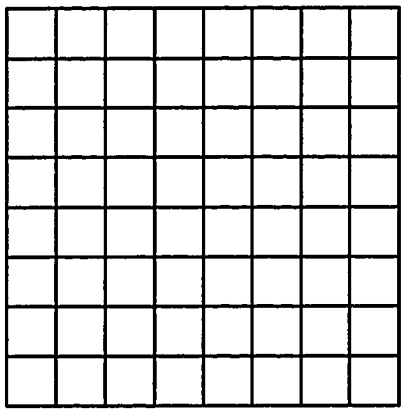
The convergence analysis of the two algorithms, the *OPMG* algorithm and the *CPMG* algorithm, which are combined in the hybrid algorithm has already been presented in Chapter 3 and Chapter 4 respectively. Therefore, for the hybrid algorithm we only give simulation results which compare its convergence rate with that of the V-cycle multigrid algorithm.



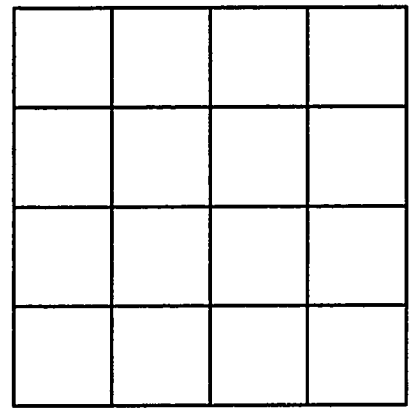
Grid G^5 (31 x 31)



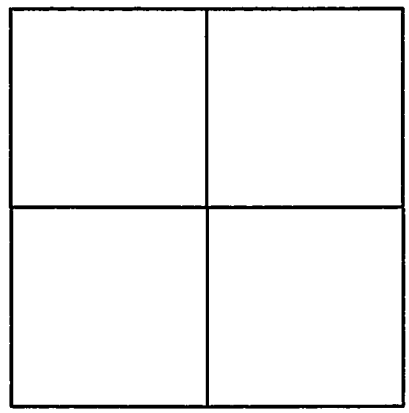
Grid G^4 (15 x 15)



Grid G^3 (7 x 7)



Grid G^2 (3 x 3)



Grid G^1 (1 x 1)

Figure 5.1 : A multigrid hierarchy of five grids

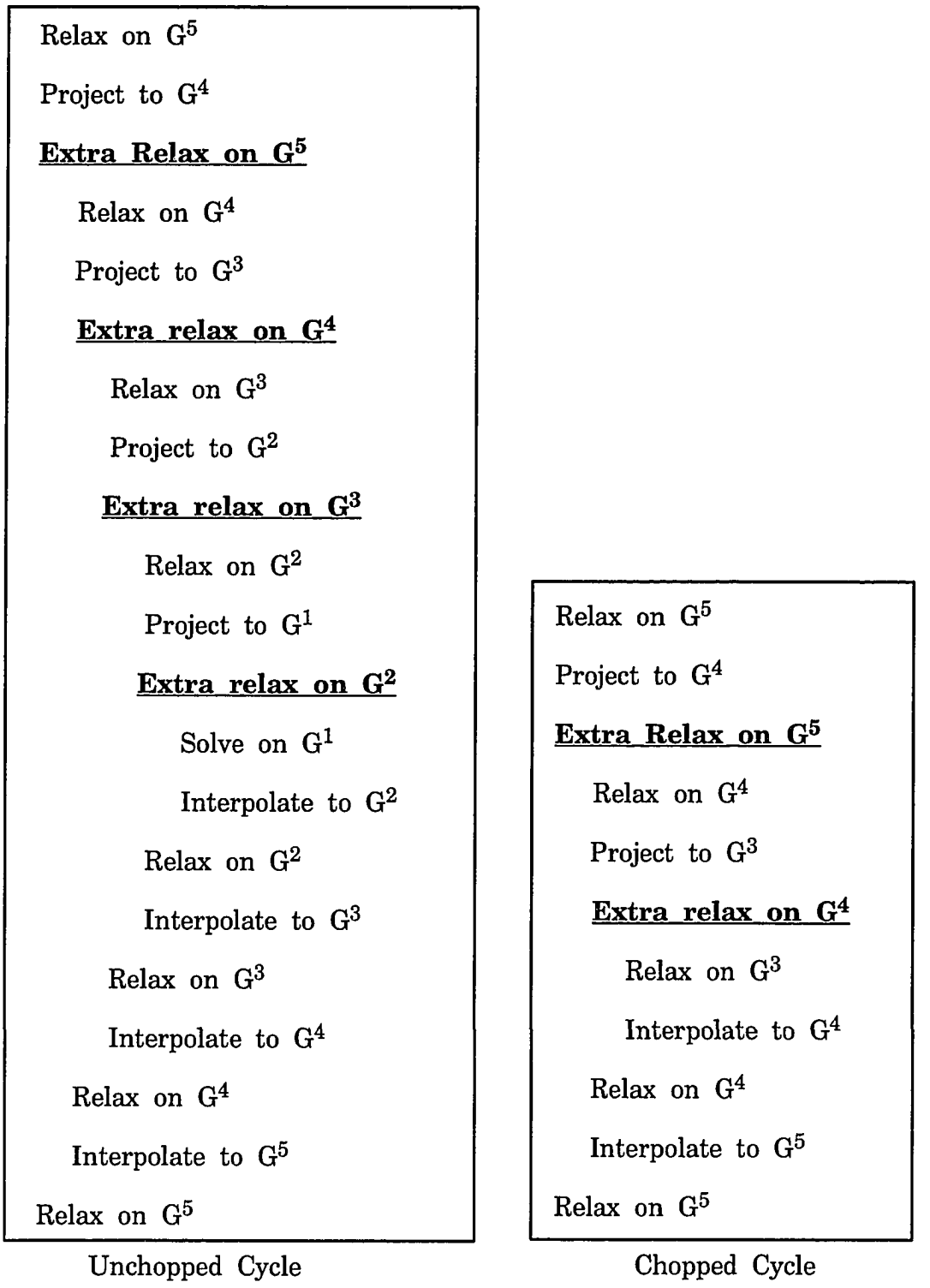


Figure 5.2 : Sequence of operations for two cycles of hybrid algorithm

The problems used for simulation are the same as those used for the *OPMG* and the *CPMG* algorithms and are described in Chapter 2. These problems require the solution of the two dimensional Poisson equation with different forcing functions. The finest grid used for the discretization of the Poisson equation has 511 points in each dimension excluding boundary points.

The results of our simulations are presented in Table 5.1 to Table 5.3. Each of these tables gives the convergence history of one problem for both the V-cycle multigrid and the hybrid multigrid algorithms. The convergence history is given in terms of the $\|.\|_2$ norm of the residual and the maximum absolute residual after each cycle of the two algorithms. In these tables, we have highlighted the entries which correspond to the same level of convergence. In other words, an entry for the hybrid algorithm which corresponds to nearly the same residual norm as that after the last cycle of the V-cycle multigrid algorithm, is highlighted.

The results in these tables show that for problems 1 and 3, the convergence rate of the hybrid algorithm is better than the convergence rate of the V-cycle multigrid algorithm. For problem 2, the convergence rate of the two algorithms is almost the same. The different behavior in this case is because the *CPMG* algorithm alone, achieves a slower convergence rate than the V-cycle multigrid algorithm for this problem (Table 4.2). In the hybrid algorithm, this poor convergence rate is compensated by the extra fine grid computation due to the *OPMG* algorithm . In all the other cases, the *CPMG* algorithm by itself achieves almost the same convergence rate as the V-cycle multigrid algorithm. Therefore, when in the hybrid

Table 5.1 : Convergence history comparison for Problem 1

No. of cycles	V-cycle residual norm	*Hybrid* residual norm	V-cycle max residual	*Hybrid* max residual
0	1.28E+0	1.28E+0	3.76E+0	3.76E+0
1	1.22E-1	6.24E-2	5.91E-1	3.76E-1
2	3.40E-2	1.36E-2	2.22E-1	9.19E-2
3	1.25E-2	3.29E-3	8.92E-2	2.48E-2
4	4.90E-3	8.68E-4	3.69E-2	6.75E-3
5	1.98E-3	2.34E-4	1.53E-2	1.88E-3
6	8.16E-4	6.43E-5	6.48E-3	5.32E-4
7	3.41E-4	**1.79E-5**	2.78E-3	1.51E-4
8	1.44E-4	5.05E-6	1.20E-3	4.29E-5
9	6.10E-5	1.45E-6	5.16E-4	1.25E-5
10	**2.60E-5**	4.44E-7	2.23E-4	3.73E-6

Table 5.2 : Convergence history comparison for Problem 2

No. of cycles	V-cycle residual norm	*Hybrid* residual norm	V-cycle max residual	*Hybrid* max residual
0	3.76E-5	3.76E-5	7.53E-5	7.53E-5
1	2.01E-5	1.99E-5	3.45E-4	4.21E-4
2	8.46E-6	1.10E-5	5.45E-5	1.73E-4
3	3.67E-6	4.72E-6	2.99E-5	6.59E-5
4	1.57E-6	2.32E-6	9.36E-6	3.02E-5
5	6.74E-7	1.01E-6	4.65E-6	1.29E-5
6	3.06E-7	5.04E-7	2.15E-6	5.72E-6
7	1.67E-7	2.45E-7	1.19E-6	2.26E-6
8	1.28E-7	1.58E-7	9.54E-7	1.19E-6
9	1.20E-7	1.26E-7	8.34E-7	8.34E-7
10	**1.18E-7**	**1.19E-7**	7.75E-7	7.75E-7

Table 5.3 : Convergence history comparison for Problem 3

No. of cycles	V-cycle residual norm	*Hybrid* residual norm	V-cycle max residual	*Hybrid* max residual
0	1.73E-2	1.73E-2	6.46E+0	6.46E+0
1	2.02E-3	1.10E-3	7.04E-1	3.36E-1
2	6.27E-4	2.50E-4	1.79E-1	6.13E-2
3	2.32E-4	6.19E-5	5.58E-2	1.21E-2
4	9.10E-5	1.61E-5	1.87E-2	2.57E-3
5	3.66E-5	4.32E-6	6.52E-3	5.80E-4
6	1.50E-5	1.17E-6	2.34E-3	1.52E-4
7	6.21E-6	**3.23E-7**	8.61E-4	4.07E-5
8	2.60E-6	8.97E-8	3.37E-4	1.11E-5
9	1.09E-6	2.51E-8	1.39E-4	3.10E-6
10	**4.64E-7**	7.06E-9	5.86E-5	7.98E-7

algorithm, it is combined with the *OPMG* algorithm, a better convergence rate is obtained.

We have also depicted the results of our simulations in the graphs of Figure 5.3 to Figure 5.5. In all the graphs, the number of cycles is plotted on the X-axis and the normalized residual norm is plotted on the Y-axis. The normalized residual norm is computed by taking the ratio of the current residual norm to the initial residual norm.

The results of our simulations are summarized in Table 5.4. Similar to Chapter 3 and Chapter 4, in this table we compare the average reduction in the residual norm per cycle for the hybrid algorithm with that of the V-cycle multigrid algorithm. The first four rows of Table 5.4 give the initial residual norm (I_{res}), the final residual norm (F_{res}), the total reduction in the residual norm and the number of cycles for the V-cycle multigrid algorithm. In row 5, we give the average reduction in the residual norm per cycle for the V-cycle multigrid algorithm. The next five rows give these quantities for the hybrid algorithm. Finally, in row 11, we give the ratio of the average reduction per cycle in the residual norm of the hybrid algorithm to that of the V-cycle multigrid algorithm. These results show that the reduction in the residual norm per cycle (β) for the hybrid algorithm is approximately 50% better than that of the V-cycle multigrid algorithm for Problems 1 and 3. For problem 2, it is the same for the two algorithms.

The results shown in these tables and graphs establish that the convergence rate of the hybrid algorithm is better than that of the V-cycle multigrid algorithm.

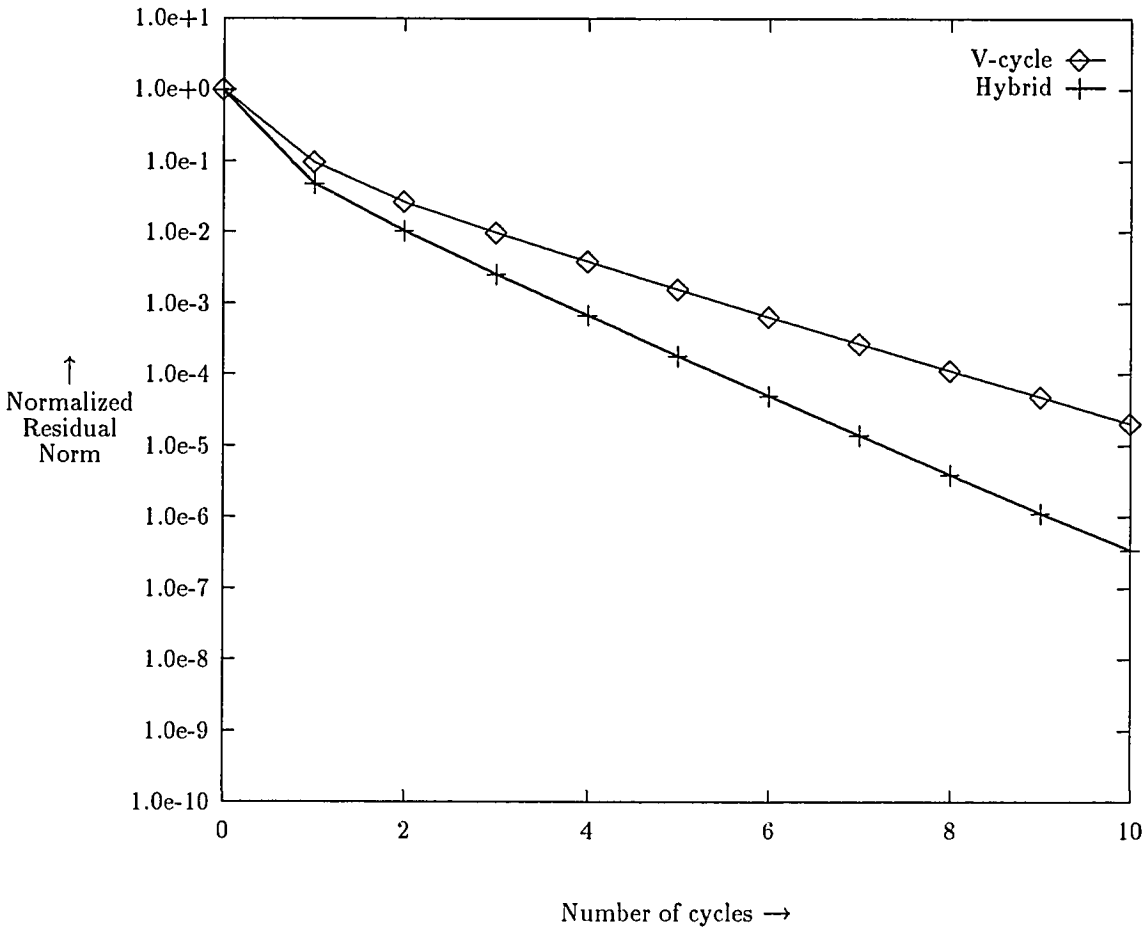


Figure 5.3 : Comparison of normalized residual norm for Problem 1

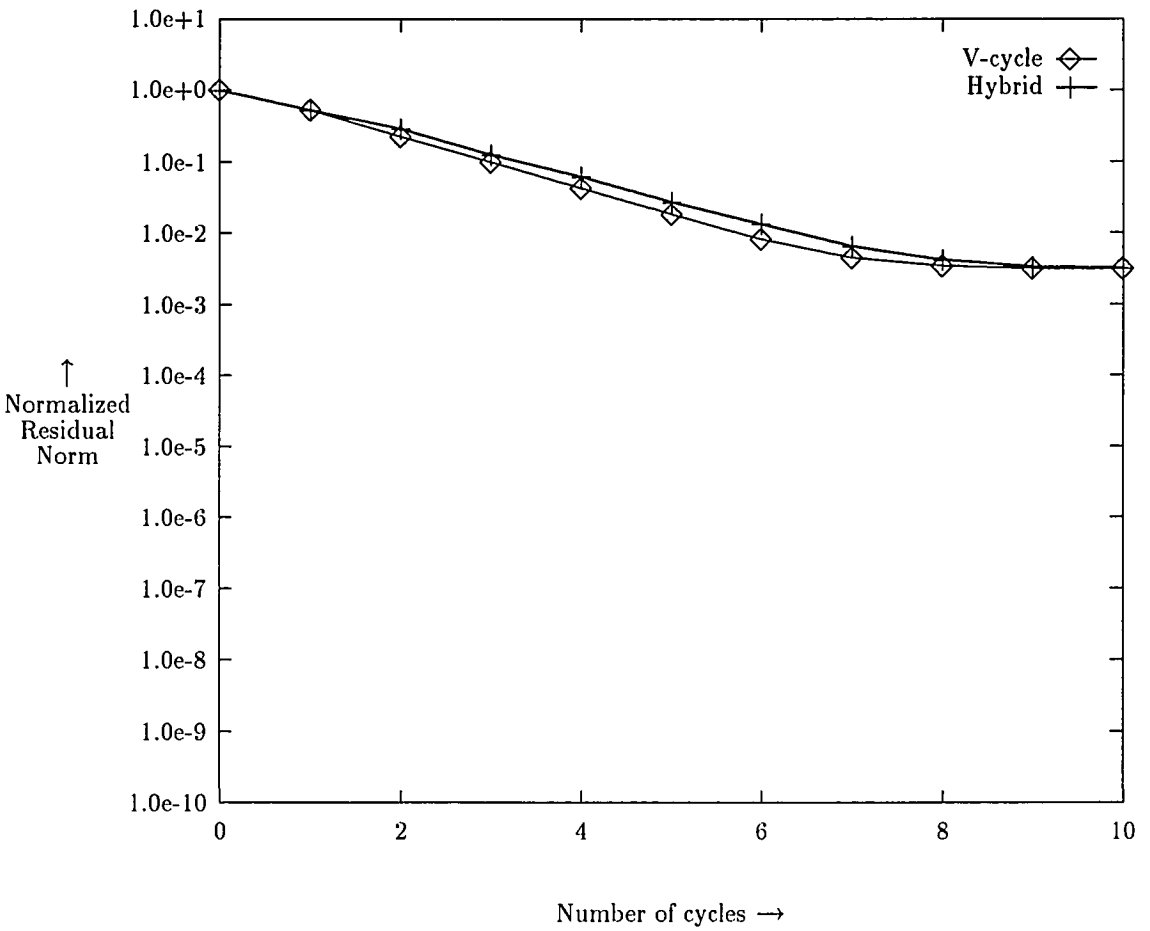


Figure 5.4 : Comparison of normalized residual norm for Problem 2

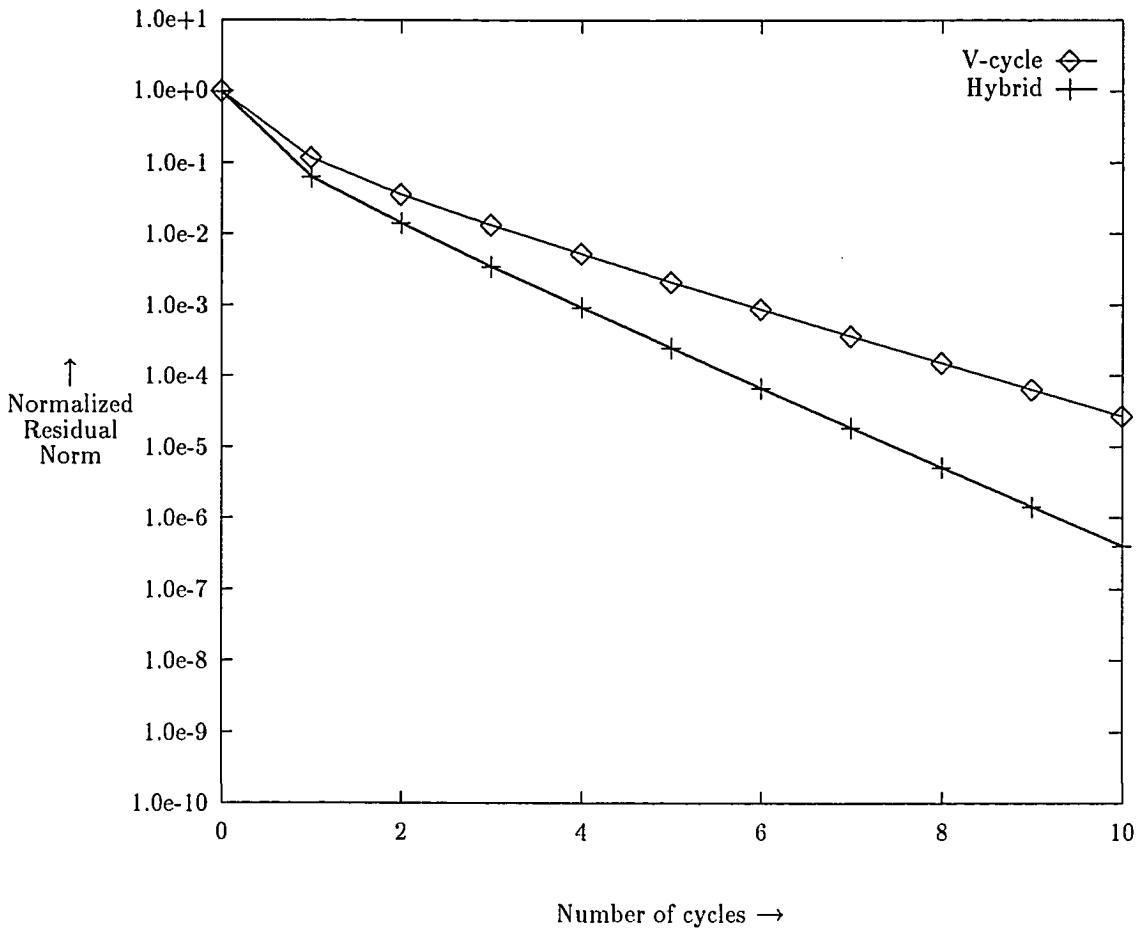


Figure 5.5 : Comparison of normalized residual norm for Problem 3

Table 5.4 : Summary of simulation results

		Problem 1	Problem 2	Problem 3
V-cycle	I_{res}	1.28	3.76E-5	1.73E-2
	F_{res}	2.60E-5	1.18E-7	4.63E-7
	$\frac{F_{res}}{I_{res}}$	2.03E-5	3.14E-3	2.68E-5
	n_{cycles}	10	10	10
	β	0.339	0.562	0.349
Hybrid	I_{res}	1.28	3.76E-5	1.73E-2
	F_{res}	4.44E-7	1.19E-7	7.06E-9
	$\frac{F_{res}}{I_{res}}$	3.47E-7	3.16E-3	4.08E-7
	n_{cycles}	10	10	10
	β	0.226	0.562	0.230
$\frac{\beta(V-cycle)}{\beta(Hybrid)}$		1.50	1.00	1.52

5.3 Implementation of Hybrid Algorithm on AMT-DAP

The parallel implementation of the hybrid algorithm follows the same strategy as that of the *OPMG* algorithm. The only difference is that instead of visiting all the grids in each cycle of the *OPMG* algorithm, we visit only half the grids in alternate cycles of the hybrid algorithm. The details of the parallel implementation of the *OPMG* algorithm are given in Chapter 3.

5.4 Performance of Hybrid Algorithm on Parallel Machines

In Section 5.2, we have shown that the hybrid algorithm achieves a better convergence rate than the V-cycle multigrid algorithm. In this section, we will show that the hybrid algorithm also has an edge over the V-cycle multigrid algorithm in terms of the parallel execution time. The results presented here correspond to the implementation of the test problems described in Chapter 2 on a 1024 processor AMT-DAP. Since the processors are arranged in a 32×32 mesh, the finest grid used for these problems has 32 grid points in each dimension. Unlike the presentation of results of the previous algorithms, here we first present detailed results of the parallel implementation. These are then followed by the evaluation of the same two performance metrics which were used for the *OPMG* and the *CPMG* algorithms.

Table 5.5 : Comparison of parallel performance for Problem 1

No. of cycles	V-cycle			*Hybrid*		
	residual norm	cycle time (sec)	cumulative time (sec)	residual norm	cycle time (sec)	cumulative time (sec)
0	1.23E+0	-	-	1.23E+0	-	-
1	1.20E-1	2.21E-2	2.21E-2	6.14E-2	2.49E-2	2.49E-2
2	3.43E-2	2.21E-2	4.42E-2	1.34E-2	1.34E-2	3.83E-2
3	1.26E-2	2.21E-2	6.63E-2	3.27E-3	2.49E-2	6.32E-2
4	4.94E-3	2.21E-2	8.84E-2	8.63E-4	1.34E-2	7.66E-2
5	2.00E-3	2.21E-2	11.05E-2	2.34E-4	2.49E-2	10.15E-2
6	8.27E-4	2.21E-2	13.26E-2	6.48E-5	1.34E-2	11.49E-2
7	3.47E-4	2.21E-2	15.47E-2	1.82E-5	2.49E-2	13.98E-2
8	1.47E-4	2.21E-2	17.68E-2	5.15E-6	1.34E-2	15.32E-2
9	6.25E-5	2.21E-2	19.89E-2	1.47E-6	2.49E-2	17.81E-2
10	2.67E-5	2.21E-2	22.10E-2	4.26E-7	1.34E-2	19.15E-2

Table 5.6 : Comparison of parallel performance for Problem 2

No. of cycles	V-cycle			*Hybrid*		
	residual norm	cycle time (sec)	cumulative time (sec)	residual norm	cycle time (sec)	cumulative time (sec)
0	9.63E-3	-	-	9.63E-3	-	-
1	4.15E-3	2.21E-2	2.21E-2	3.50E-3	2.49E-2	2.49E-2
2	1.60E-3	2.21E-2	4.42E-2	1.70E-3	1.34E-2	3.83E-2
3	6.22E-4	2.21E-2	6.63E-2	6.06E-4	2.49E-2	6.32E-2
4	2.44E-4	2.21E-2	8.84E-2	2.90E-4	1.34E-2	7.66E-2
5	9.64E-5	2.21E-2	11.05E-2	1.03E-4	2.49E-2	10.15E-2
6	3.86E-5	2.21E-2	13.26E-2	4.90E-5	1.34E-2	11.49E-2
7	1.57E-5	2.21E-2	15.47E-2	1.73E-5	2.49E-2	13.98E-2
8	6.56E-6	2.21E-2	17.68E-2	8.27E-6	1.34E-2	15.32E-2
9	2.83E-6	2.21E-2	19.89E-2	3.03E-6	2.49E-2	17.81E-2
10	1.36E-6	2.21E-2	22.10E-2	1.53E-6	1.34E-2	19.15E-2

Table 5.7 : Comparison of parallel performance for Problem 3

	V-cycle			*Hybrid*		
No. of cycles	residual norm	cycle time (sec)	cumulative time (sec)	residual norm	cycle time (sec)	cumulative time (sec)
0	2.46E+0	-	-	2.46E+0	-	-
1	2.40E-1	2.21E-2	2.21E-2	1.23E-1	2.49E-2	2.49E-2
2	6.86E-2	2.21E-2	4.42E-2	2.68E-2	1.34E-2	3.83E-2
3	2.52E-2	2.21E-2	6.63E-2	6.53E-3	2.49E-2	6.32E-2
4	9.88E-3	2.21E-2	8.84E-2	1.72E-3	1.34E-2	7.66E-2
5	4.00E-3	2.21E-2	11.05E-2	4.68E-4	2.49E-2	10.15E-2
6	1.65E-3	2.21E-2	13.26E-2	1.29E-4	1.34E-2	11.49E-2
7	6.93E-4	2.21E-2	15.47E-2	2.62E-5	2.49E-2	13.98E-2
8	2.93E-4	2.21E-2	17.68E-2	1.02E-5	1.34E-2	15.32E-2
9	1.25E-4	2.21E-2	19.89E-2	2.96E-6	2.49E-2	17.81E-2
10	5.34E-5	2.21E-2	22.10E-2	1.10E-6	1.34E-2	19.15E-2

The results of our implementation are presented in Table 5.5 to Table 5.7 for test problem 1 to test problem 3 respectively. The first three columns of these tables give the residual norm, the parallel execution time for the current cycle and the cumulative parallel execution time, for the V-cycle multigrid algorithm. Similarly, the next three columns give these three quantities for the hybrid algorithm. These results indicate that for problems 1 and 3, the reduction in the residual norm is more for the hybrid algorithm in comparison to the V-cycle multigrid algorithm. For problem 2, which represents the worst case, the reduction in the residual norm for the two algorithms is almost the same. Also, for all the problems, the cumulative parallel execution time for the hybrid algorithm is less than the cumulative parallel execution time for the V-cycle multigrid algorithm. Notice that the parallel execution time for the unchopped cycles of the hybrid algorithm (odd numbered cycles), is slightly more than the cycle time for the V-cycle multigrid algorithm because of the overheads caused by the extra fine grid computation of the *OPMG* algorithm. This extra time in an unchopped cycle of the hybrid algorithm is compensated by the next cycle which is chopped and takes much less time than the corresponding cycle of the V-cycle multigrid algorithm.

We also compare the performance of the two algorithms using the performance metrics defined in Chapter 3. The first metric is the average reduction in the residual norm per parallel time unit (ptu) . The results of our comparison using this metric are shown in Table 5.8. Similar to earlier Chapters, the first four rows of this table give the average reduction in the residual norm per cycle (β), the parallel execution

time per cycle in seconds, the parallel execution time per cycle in *ptu* and the average reduction in the residual norm per *ptu* (γ) for the V-cycle multigrid algorithm. The next four rows of the table give the same four quantities for the hybrid algorithm. In the last row, we give the ratio of $\gamma(V-cycle)$ to $\gamma(hybrid)$. The results of this table show that the hybrid algorithm achieves a better performance both in execution time and in convergence per cycle. The combination of these two factors shows an advantage of approximately 90% for the hybrid algorithm in case of problems 1 and 3. For problem 2, advantage is only 14% because the convergence rate per cycle is almost the same for the two algorithms.

The results of comparison using the second metric are shown in Table 5.9. Here, in the first two rows, we give the number of cycles required by the two algorithms to reach the same level of convergence (H_{cycles} and V_{cycles}). In the next two rows, we give the total execution time (in seconds), required by the two algorithms to reach the same level of convergence (H_{total} and V_{total}). Finally, in row 5, we give the speed-up, as defined in Chapter 3.

The results in this table clearly establish the superior performance of the hybrid algorithm compared to the V-cycle multigrid algorithm on massively parallel machines for all the three test problems. The hybrid algorithm for problems 1 and 3 has a speed-up of approximately 1.6 (60%) over the V-cycle multigrid algorithm. For problem 2, the speed-up is approximately 1.15 (15%). The not so good improvement in the performance of the hybrid algorithm for problem 2 is due to the poor convergence rate of the *CPMG* algorithm for this problem. It should be noted

Table 5.8 : Parallel implementation results for the Hybrid algorithm (Performance metric 1)

		Problem 1	Problem 2	Problem 3
V-cycle	β	0.342	0.412	0.342
	Execution Time (sec.)	2.21E-2	2.21E-2	2.21E-2
	Execution Time (ptu)	1.0	1.0	1.0
	γ	0.342	0.412	0.342
Hybrid	β	0.225	0.417	0.231
	Execution Time (sec.)	1.91E-2	1.91E-2	1.91E-2
	Execution Time (ptu)	0.86	0.86	0.86
	γ	0.177	0.362	0.182
$\frac{\gamma(V-cycle)}{\gamma(hybrid)}$		1.93	1.14	1.88

Table 5.9 : Parallel implementation results for the Hybrid algorithm

(Performance metric 2)

	Problem 1	Problem 2	Problem 3
H_{cycles}	7	10	7
V_{cycles}	10	10	10
H_{total}	1.39E-1	1.91E-1	1.39E-1
V_{total}	2.21E-1	2.21E-1	2.21E-1
$speed - up$	1.59	1.16	1.59

that problem 2 is the worst case, where all the error is concentrated in the lowest frequency mode.

5.5 Conclusions

In this chapter, we have presented the hybrid multigrid algorithm. The hybrid algorithm combines the features of the *OPMG* algorithm and the *CPMG* algorithm. The advantage of combining the two algorithms reflects in the convergence rate as well as the parallel execution time. The results of our parallel implementation of the hybrid algorithm on a massively parallel machine, the AMT-DAP, show an advantage of approximately 60% in execution time over the V-cycle multigrid algorithm. The improvement in performance for the hybrid algorithm is better than both the *OPMG* and the *CPMG* algorithms individually.

Chapter 6

Conclusions

6.1 Conclusions

In this dissertation, we have addressed the problems in the implementation of the multigrid algorithm on massively parallel machines. The two major problems are low processor utilization and high communication overheads. To address these problems we have proposed two algorithms, the *Overlap Parallel Multigrid* algorithm and the *Chopped Parallel Multigrid* algorithm. We have also presented a hybrid algorithm, which combines the advantages of both the algorithms. All the three algorithms achieve a better performance on massively parallel machines in comparison to the V-cycle multigrid algorithm. The advantage is shown in terms of reduced parallel execution time, which is achieved by improving processor utilization and reducing communication overheads.

The *OPMG* algorithm obtains a better convergence rate than the V-cycle multigrid algorithm by doing extra computation on each grid of the multigrid hierarchy. The extra computation is accommodated on the unutilized processors of the coarse grids. We have given an implementation of the *OPMG* algorithm on a massively parallel SIMD machine AMT-DAP/510. In our implementation, we use a complex communication and masking scheme which results in approximately 10% overhead. We have compared the performance of the *OPMG* algorithm with the V-cycle multigrid algorithm. This comparison shows that the *OPMG* algorithm achieves approximately 25% to 35% better average convergence rate per parallel time unit than the V-cycle multigrid algorithm. In terms of the total parallel execution time, the *OPMG* algorithm obtains approximately 30% speed-up over the V-cycle multigrid algorithm.

The *CPMG* algorithm reduces the average parallel execution time per cycle while keeping the convergence rate the same, in comparison to the V-cycle multigrid algorithm. This is achieved by reducing the amount of computational work done on the coarse grids in comparison to the fine grids. Since the problems of low processor utilization and high communication overheads are due to the computation on the coarse grids, the relative reduction in the computational work on these grids effectively addresses these problems. Using analytical and simulation results, we have shown that this reduction in the computational work on the coarse grids does not affect the convergence rate significantly for a number of problems. The *CPMG* algorithm achieves approximately 40% better convergence rate per parallel time unit

in comparison to the V-cycle multigrid algorithm. For test problem 2, the *CPMG* algorithm's performance is slightly worse than the V-cycle multigrid algorithm. This is because the initial error for test problem 2 contains only the lowest frequency mode, for which the convergence of the *CPMG* algorithm is worse than that of the V-cycle multigrid algorithm. Comparison of the *CPMG* algorithm and the V-cycle multigrid algorithm in terms of the total parallel execution time shows similar improvements.

The third algorithm presented in this dissertation, the hybrid algorithm, combines the features of the *OPMG* algorithm and the *CPMG* algorithm. By combining these two algorithms, the hybrid algorithm achieves a better convergence rate and also reduces the average parallel execution time per cycle. The results from our parallel implementation of the hybrid algorithm show that the average convergence rate per parallel time unit is approximately 90% better than that of the V-cycle multigrid algorithm. The hybrid algorithm also gives approximately 60% advantage in the total parallel execution time. For test problem 2, the advantage of the hybrid algorithm is not very high. This is because, for this problem, the reduction in the computational work on the coarse grids degrades the convergence rate. Note, that this problem is the worst case where all the error is contained in the lowest frequency mode.

6.2 Future Work

With the advancement in parallel computer technology over the recent years, it has become feasible to construct massively parallel machines consisting of thousands of processors. The algorithms presented in this dissertation can significantly reduce the computational cost of solving very large scientific problems involving partial differential equations, on these machines. We would like to implement the work done in this dissertation on the state of art parallel machines for solving these scientific problems.

All the work done in this dissertation pertains to solving two dimensional problems. An implementation of the proposed algorithms for three dimensional cases will be of great importance. Finally, we would like to make use of this work for addressing the problems in the parallel implementation of the multigrid algorithm on various parallel architectures such as distributed and shared memory MIMD architectures.

References

[1] A. Brandt, "Multi-level Adaptive Solutions to Boundary Value Problems," *Mathematics of Computation*, 31, 1977, pp. 333-390.

[2] W. Briggs, "A Multigrid Tutorial," *Society for Industrial and Applied Mathematics,* Philadelphia, 1987.

[3] T. Chan and R. Schreiber, "Parallel Networks for Multigrid Algorithms : Architecture and Complexity," *SIAM J. Sci. Stat. Computing*, Vol. 6, No. 3, July 1985, pp. 698-711.

[4] T. Chan, and Y. Saad, "Multigrid Algorithms on the Hypercube Multiprocessors," *IEEE Transactions on Computers*, Vol. C-35, No. 11, Nov. 1986, pp. 969-977.

[5] T. Chan and R. Tuminaro, "A Survey of Parallel Multigrid Algorithms", *Parallel Computers and Their Impact on Mechanics*, A. K. Noor (ed.), AMD-1986, pp. 155-170.

[6] T. Chan and R. Tuminaro, "Design and Implementation of Parallel Multigrid Algorithms," *Proceedings of the Third Copper Mountain Conference on Multigrid Methods*, S. McCormick (ed.), Marcel Dekker, NY, 1987.

[7] T. Chan and R. Tuminaro, "Analysis of a Parallel Multigrid Algorithm," *Proceedings of the Fourth Copper Mountain Conference on Multigrid Meth-*

ods, J. Mandel and S. McCormick (eds.), SIAM, Philadelphia, 1989, pp. 66-86.

[8] N. H. Decker, "On the Parallel Efficiency of the Fredrickson-McBryan Multigrid ", ICASE Report No. 90-17, *Institute for Computer Applications in Science and Engineering*, Hampton, VA, February 1990.

[9] N. Decker, "Note of the Parallel Efficiency of the Frederickson-McBryan Multigrid Algorithm," *SIAM J. Sci. Stat. Comput.*, Vol. 12, No. 1, 1991, pp. 208-220.

[10] R. P. Federenko, "A Relaxation Method for Solving Elliptic Partial Differential Equations", *USSR Comp. Math. and Math. Physics,* Vol.1, 1961, pp. 922-927. (In Russian)

[11] R. P. Federenko, "The Speed of Convergence on One Iteration Process", *USSR Comp. Math. and Math. Physics,* Vol.4, 1961, pp.227-235. (In Russian)

[12] P. Frederickson and O. McBryan, "Parallel Superconvergent Multigrid," *Proceedings of the Third Copper Mountain Conference on Multigrid Methods,* S. McCormick (ed), Marcel Dekker, NY, 1989.

[13] P. Frederickson and O. McBryan, "Normalized Convergence Rates for the PSMG Methods," *SIAM J. Sci. Stat. Comput.*, Vol. 12, No. 1, 1991, pp. 221-229.

[14] *Fortran-Plus Language (man002.02)*, Active Memory Technology, 1988.

[15] D. Gannon and J. Van Rosendale, "On the Structure of Parallelism in a Highly Concurrent PDE solver," *Journal of Parallel and Distributed Computing*, Vol. 3, 1986, pp. 106-135.

[16] G. Golub and C. Van Loan, "Matrix Computations," The John Hopkins University Press, Baltimore, Maryland, 1983.

[17] A. Greenbaum, "A Multigrid Method for Multiprocessors," *Proceedings of the Second Copper Mountain Conference of Multigrid Methods*, S. McCormick (ed.), Appl. Math. and Computation, Vol. 19, 1986, pp. 75-88.

[18] C.E. Grosch, "Performance Analysis of Poisson Solvers on Array Computers," *Supercomputers*, Vol. 2, C.R. Jesshope and R.W. Hockney, eds. Infotech Intl., 1979, pp. 147-181.

[19] W. Hackbusch and U. Totenberg, *Multigrid methods,* Springer-Verlag, Berlin, 1982.

[20] A. Jameson and S. Yoon, "Multigrid Solution of the Euler Equations Using Implicit Schemes," *AIAA Journal*, Vol. 24, No. 11, Nov. 1986.

[21] A. Jameson and S. Yoon, "Lower-Upper Implicit Schemes with Multiple Grids for the Euler Equations," *AIAA Journal*, Vol. 25, No. 7, July 1987.

[22] O. Kolp and H. Mierendroff, "Efficient Multigrid Algorithms for Locally Constrained Parallel Systems," *Proceedings of the Second Copper Moun-*

tain Conference of Multigrid Methods, S. McCormick (ed.), Appl. Math. and Computation, Vol. 19, 1986, pp. 169-200.

[23] C. Lin, W. Proskurowski and J. Gaudiot, "A Parallel Multigrid Method for Data-Driven Multiprocessor Systems," *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, J. Mandel and S. McCormick (eds.), SIAM, Philadelphia, 1989, pp. 299-318.

[24] O. McBryan, "The Connection Machine : PDE Solution on 65536 Processors," *Parallel Computing*, Vol. 9, 1988/89, pp. 1-24.

[25] R. Peyret and T. D. Taylor, "Computational Methods for Fluid Flows", Springer-Verlag, New York, 1983.

[26] S. F. Reddaway, "DAP - a Distributed Array Processor," *First International Symposium on Computer Architecture*, 1973, pp. 61-65.

[27] T. Siikonen and J. Hoffren, "Multigrid Solution Method for Euler Equations", Report No. A-10, Series A, Laboratory of Aerodynamics, Helsinki University of Technology, 1989.

[28] H. S. Stone, "High Performance Computer Architecture", Addison-Wesley Publishing Company, Reading, Massachussetts, 1987.

[29] K. Stuben and U. Trottenberg, "Multigrid Methods : Model Problem Analysis and Application," *Multigrid Methods*, W. Hackbusch and U. Trottenberg (eds.), Springer-Verlag, N.Y., 1982, pp. 1-176.

[30] J. Swisshelm, G. Johnson and S. Kumar, "Parallel Computation of Euler and Navier-Stokes Flows," *Proceedings of the Second Copper Mountain Conference of Multigrid Methods*, S. McCormick (ed.), Appl. Math. and Computation, Vol. 19, 1986, pp. 321-331.

[31] R. Tuminaro, "A highly Parallel Multigrid-Like Method for the Solution of the Euler Equations," *SIAM J. Sci. Stat. Comput.*, Vol. 13, No. 1, 1992, pp. 88-100.

[32] R. S. Varga, "Matrix Iterative Analysis," Prentice Hall, Englewood Cliffs, 1962.

Autobiographical Statement

Satyanarayan Gupta

I was born on August 1, 1961 in Jaipur, India. I stayed in India from my birth until 1988, when I joined the Computer Science Department, Old Dominion University, Norfolk, VA. I am married to Madhurima, who is also a Doctoral student in the Computer Science Department.

I have the following two degrees :

- B. E., Electronics and Electrical Engineering, Birla Institute of Technology and Science, Pilani, India, June 1983.
- M. Tech., Computer Technology, Indian Institute of Technology, New Delhi, India, June 1988.

When I was in India, I worked as Senior Research Assistant from 1983 to 1985 and Senior Scientific Officer from 1985 to 1988 at the Indian Institute of Technology, New Delhi, India. I have also worked as a graduate fellow at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA, from May 1991 to July 1992.

The following is a list of my published articles :

1. R. Das, D. Mavriplis, J. Saltz, S. Gupta and R. Ponnusamy, " The Design and Implementation of a Parallel Unstructured Euler Solver using Software Primitives," *AIAA Paper Number 92-0562*, 1992.

2. M. Zubair, S.N. Gupta and C.E. Grosch, "A Variable Precision Approach to Speed-Up Iterative Schemes on Fine Grained Parallel Machines", *Parallel Computing* (To Appear).

3. S.N. Gupta, M. Zubair and C.E. Grosch, "Implementation of an Oversize Neural Network on DAP-510," *Fifth International Parallel Processing Symposium*, Anheim, California, 1991.

4. M. Zubair and S.N. Gupta , "Embeddings of a boolean cube", *BIT 30*, pp. 245-256, 1990

5. S.N. Gupta, M. Zubair and C.E. Grosch, "Visualization : An Aid to Design and Understand Neural Networks in Parallel Environment," *Proceedings of The Fifth Distributed Memory Computing Conference,* Charleston, South Carolina, 1990.

6. C.E. Grosch, M. Ghose, S.N. Gupta, T.L. Jackson and M. Zubair, "Massively Parallel Computation of the Euler Equations," *Proceedings of The Fifth Distributed Memory Computing Conference,* Charleston, South Carolina, 1990.

7. S. N. Gupta, M. Zubair and C.E. Grosch, "Simulation of Neural Networks on a Massively Parallel Computer (DAP-510) Using Sparse Matrix Techniques," *Proceedings of The Third Symposium on The Frontiers of Massively Parallel Computation,* College Park, Maryland, 1990.

8. S.N. Gupta, M. Zubair and C.E. Grosch, "A study of sorting algorithms of a

mesh connected computer (DAP 510)", *SIAM Conference on Parallel Computing*, Chicago, Illinois, 1989.

9. Sudhir Aggarwal, S.N. Gupta and A.B. Bhattacharya, "Design Rule Checking and Expert Systems", *International Workshop on VLSI Design*, Bangalore, India, 1988.

10. Anshul Kumar, S.N. Gupta, S. Prasad and Jyotsana Bahl," Synthesis of floorplan for control part in CADIT system", *International Workshop on VLSI Design*, Bangalore, India, 1988.

11. S.N. Gupta, M. Zubair and C.E. Grosch, "A Multigrid Algorithm for Massively Parallel Computers : CPMG", Submitted for publication to the *Journal of Scientific Computing*, 1992.

12. S.N. Gupta, M. Zubair and C.E. Grosch, "An Overlap Parallel Multigrid Algorithm", *Under preparation*, 1992.

13. S.N. Gupta and J. Saltz, "Runtime Support for Unstructured Problems on Massively Parallel SIMD Machines,", *Under Preparation*, 1992.